University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Błażej Bogumił Osiński

Efficient Methods for Machine Learning in Sequential Decision Making

PhD dissertation
in COMPUTER SCIENCE

Supervisor:
dr hab. Piotr Miłoś
Polish Academy of Sciences

Warsaw, June 2023

Author's declaration:

I hereby declare that this dissertation is my own work.

Date: _____ Signature: _____

mgr Błażej Osiński

Supervisor's declaration:

The dissertation is ready to be reviewed.

Date: _____ Signature: _____

dr hab. Piotr Miłoś, prof. IM PAN

Tytuł w języku polskim

Wydajne metody uczenia maszynowego dla problemów sekwencyjnego podejmowania decyzji

Streszczenie

W niniejszej pracy badane są efektywne i wydajne rozwiązania dla problemów sekwencyjnego podejmowania decyzji - klasy problemów, w których agent wchodzi w interakcję ze środowiskiem, wykonując serię akcji, w celu osiągnięcia określonego celu. Jednym z głównych wyzwań w tej dziedzinie jest koszt związany ze zbieraniem danych, który jest często przeszkodą w szerszym zastosowaniu metod opartych o uczenie maszynowe. Podejmując to wyzwanie, niniejsza praca bada trzy strategie mające na celu zmniejszenie kosztów uzyskania danych: zwiększanie efektywności wykorzystania danych przez algorytmy, użycie tańszych źródeł danych oraz posłużenie się wiedzą zakodowaną w wytrenowanych modelach uczenia maszynowego.

W poszukiwaniu zwiększania efektywności wykorzystania danych, zastosowano uczenie ze wzmocnieniem oparte o model. Zaproponowany algorytm, SimPLe, jest pierwszym skutecznym zastosowaniem tej klasy metod do gier Atari. Ustanowił on najwyższe wyniki pod względem efektywności wykorzystania danych i wywołał większe zainteresowanie ewaluacją algorytmów w reżimie małej ilości danych.

Druga prezentowana strategia, korzystanie z tańszych źródeł danych, została wykorzystana w kontekście samochodów autonomicznych. Pokazana została możliwość użycia danych z symulatora do wytrenowania polityki mogącej prowadzić samochód w świecie rzeczywistym. Zbadana została też możliwość wykorzystania uczenia przez imitację, które potrzebuje wyłącznie statycznego zbioru danych trajektorii eksperta, bez potrzeby kosztownego i ryzykownego uruchamiania własnej polityki. Wychodząc od modułu planującego opartego o uczenie maszynowe o nazwie ChauffeurNet, zaproponowano cztery usprawnienia w zakresie jakości danych, treningu modelu, walidacji i uruchamiania w świecie rzeczywistym.

Na koniec, zaprezentowano możliwość wykorzystania w robotyce wiedzy zawartej w wytrenowanych modelach, takich jak duże modele językowe. Zaproponowana metoda o nazwie LM-Nav wyróżnia się zdolnością do wykonywania w świecie rzeczywistym instrukcji w języku naturalnym. Nie potrzebuje w tym celu trenować ani dopasowywać wykorzystywanych modeli.

Podsumowując, niniejsza rozprawa prezentuje nowatorskie i skuteczne metody rozwiązywania problemów sekwencyjnego podejmowania decyzji. Mam nadzieję, że przyczyni się do szerszego wykorzystania uczenia maszynowego w tej klasie problemów w świecie rzeczywistym.

Słowa kluczowe

uczenie ze wzmocnienie, uczenie w oparciu o model, transfer symulator-rzeczywistość, uczenie przez imitację, modele podstawowe, autonomiczne samochody

Klasyfikacja tematyczna ACM

Computing methodologies → Machine learning → Learning paradigms → Reinforcement learning → Sequential decision making

Abstract

This dissertation explores efficient and cost-effective solutions for sequential decision-making problems – a class of problems where an agent continuously interacts with an environment, making a series of actions, with the aim to accomplish a certain objective. One of the fundamental challenges in this area is the significant expense associated with gathering data, often hindering the broader adoption of methods based on machine learning. In response, this thesis explores three overarching strategies to mitigate these costs: enhancing the sample efficiency of algorithms, utilizing more affordable data sources, and leveraging knowledge already encoded in pre-trained machine learning models.

In the quest for improved sample efficiency, the focus is on model-based reinforcement learning. The proposed SimPLe algorithm is the first successful application of these methods to Atari games. It established state-of-the-art results in sample efficiency and sparked interest in evaluating algorithms in the low-data regime.

The second presented strategy, of utilizing more affordable data sources, is explored in the context of autonomous driving. The possibility of using simulated data for training a driving policy suitable for real-world deployment is demonstrated. Furthermore, the potential of imitation learning is explored, which relies solely on offline data, promoting a less expensive and safer data acquisition approach. From a starting point of a machine learning-based planner akin to ChauffeurNet, four distinct enhancements to data collection, model training, validation, and deployment are proposed.

Finally, the potential of capitalizing on the knowledge embedded in pre-trained models, such as large language models, is unveiled in the context of robotics. The method, dubbed LM-Nav (Large Model Navigation), stands out for its ability to follow natural language instructions in real-world environments, all without the need for supplementary training or fine-tuning. This demonstrates a unique approach to utilize pre-existing models to manage complex tasks efficiently.

Collectively, this dissertation presents novel and effective approaches for sequential decision-making problems, paving the way for the broader adoption of machine learning methods for this class of problems in the real world.

Keywords

reinforcement learning, model-based reinforcement learning, sim-to-real transfer, imitation learning, foundation model, autonomous driving

ACM Computing Classification

Computing methodologies → Machine learning → Learning paradigms → Reinforcement learning → Sequential decision making

I would like to dedicate this dissertation
to the three most important women in my life:
to my wife, Albertyna,
my mother, Aniela,
and my daugher, Wanda.

Contents

List of publications

The body of my doctoral thesis comprises seven publications enumerated below. A star denotes lead authors with equal contributions.

1. Ł. Kaiser*, M. Babaeizadeh*, P Miłos*, **B. Osiński***, R. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, H. Michalewski
   Model-based Reinforcement Learning for Atari
   spotlight at Eighth International Conference on Learning Representations, 2020

   My contributions include the implementation of the initial version of the model-based RL algorithm and world model together with Piotr Miłoś, the idea of random starts, the execution of many experiments, in particular, related to policy training, and co-writing the manuscript.
   I estimate this to be 20% of the work.

2. **B. Osiński**, A. Jakubowski, P. Zięcina, P. Miłoś, C. Galias, S. Homoceanu, H. Michalewski
   Simulation-based reinforcement learning for real-world autonomous driving
   IEEE International Conference on Robotics and Automation, 2020

   My contributions include designing and executing a part of the experiments, supervising the real-world experimental verification, coordinating the work of software engineers on the team, and co-writing the manuscript.
   I estimate this to be 35% of the work.

3. D. Shah*, **B. Osiński***, B. Ichter, S. Levine
   LM-Nav: Robotic navigation with large pre-trained models of language, vision, and action,
   Conference on Robot Learning (CoRL), 2022

   My contributions include writing code for language processing, landmarks grounding, and graph search. I also co-designed the experiments and specified the details of the probabilistic interpretation of the search algorithm. Finally, I wrote a big part of the manuscript.
   I estimate this to be 50% of the work.

4. L. Chen*, L. Platinsky*, S. Speichert*, **B. Osiński**, O. Scheel, Y. Ye, H. Grimmett, L. Del Pero, P. Ondruska
   What data do we need for training an av motion planner?
   IEEE International Conference on Robotics and Automation (ICRA), 2021

   My contributions include co-writing the manuscript.
   I estimate this to be 5% of the work.

5. L. Bergamini*, Y. Ye*, O. Scheel, L. Chen, C. Hu, L. Del Pero, **B. Osiński**, H. Grimmett, P. Ondruska
SimNet: Learning reactive self-driving simulations from real-world observations,
IEEE International Conference on Robotics and Automation (ICRA), 2021

   My contributions include proposing a narrative of the paper highlighting the impact of SimNet on ML planner verification, investigating of the causal confusion problem, and co-writing the manuscript.
   I estimate this to be 10% of the work.

6. O. Scheel, L. Bergamini, M. Wołczyk, **B. Osiński**, P. Ondruska
Urban driver: Learning to drive from real-world demonstrations using policy gradients,
Conference on Robot Learning (CoRL), 2021

   My contributions include co-supervising the experiment design and execution and co-writing the manuscript.
   I estimate this to be 15% of the work.

7. M. Vitelli*, Y. Chang*, Y. Ye*, A. Ferreira, M. Wołczyk, **B. Osiński**, M. Niendorf, H. Grimmett, Q. Huang, A. Jain, P. Ondruska
SafetyNet: Safe planning for real-world self-driving vehicles using machine-learned policies,
IEEE International Conference on Robotics and Automation (ICRA), 2022

   My contributions include conducting experiments on the ML planner and co-writing the manuscript.
   I estimate this to be 5% of the work.

During my doctoral studies, I have also published three more articles listed below. They are not included in the thesis because they are not directly related to the main topic of my research.

1. Ł. Kidziński, S. P. Mohanty, C. F. Ong, Z. Huang, S. Zhou, A. Pechenko, A. Stelmaszczyk, P. Jarosik, M. Pavlov, S. Kolesnikov, S. Plis, Z. Chen, Z. Zhang, J. Chen, J. Shi, Z. Zheng, C. Yuan, Z. Lin, H. Michalewski, P. Miłoś, **B. Osiński**, A. Melnik, M. Schilling, H. Ritter, S. F. Carroll, J. Hicks, S. Levine, M. Salathé, S. Delp
Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments,
The NIPS'17 Competition: Building Intelligent Systems. Springer International Publishing, 2018.

2. M. Zawalski, **B. Osiński**, H. Michalewski, P. Miłoś
Off-Policy Correction For Multi-Agent Reinforcement Learning,
21st International Conference on Autonomous Agents and Multiagent Systems, 2022.

3. **B. Osiński**, P. Miłoś, A. Jakubowski, P. Zięcina, M. Martyniak, C. Galias, A. Breuer, S. Homoceanu, H. Michalewski
CARLA Real Traffic Scenarios–novel training ground and benchmark for autonomous driving
Autonomous Driving Workshop, NeurIPS 2020

Chapter 1

Introduction

In 1988, the year when I was born, Hans Moravec wrote [43]:

> *It is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility.*

Even after the passage of over three decades, the essence of Moravec's paradox, as it is commonly referred to, remains true. Over the last few decades, artifical intelligence managed to ascend what were perceived as pinnacles of human intelligence. Starting with the victory of Deep Blue over Garry Kasparov in 1997 [12], through the victory of AlphaGo over Lee Sedol in 2016 [58], to the victory of AlphaStar over professional StarCraft players in 2019 [63], the artificial intelligence has been able to defeat the best human players in various games. The advent of large language models [16, 10] has shown that the progress is not limited to competitive games, but can also be achieved in more open-ended domains. The recently introduced GPT-4 [46] was evaluated on a wide range of exams, including SAT (college admission), Uniform Bar Examination (law), and USABO Semifinal (biology). In the aforementioned exams, GPT-4 performed above the 88th percentile of human test takers. Notably, it surpassed the passing threshold for all UBE jurisdictions with a considerable margin [33], i.e. it would be qualified to represent a client in a court of law.

Despite these successes, the progress of artificial intelligence when interacting with the physical world is less remarkable. A vivid example of this is manipulation using multifingered robotic hand [38]. Despite 30 years of research, robots are still much less adept at using hands than average human children. A recent, celebrated example of progress was enabling a robot to solve a Rubik's cube [2]. However, the robot required 4 minutes to solve it, and a human master can do it in slightly more than 6 seconds (also using one hand) [52]. Drawing a parallel to the context of examinations, it is highly probable that current robotic systems would struggle to sufficiently manipulate a pen to complete a written exam within the designated time frame.

The comparatively slower advancements in the realm of physical interactions can be largely attributed to the massive data requirements of contemporary machine learning methodologies. Collecting huge datasets beyond the game-based environments and simulators can incur substantial costs. The data complexity can escalate further when we shift focus from perception-based tasks, such as image object recognition, towards the automation of control tasks like vehicular driving. Many contemporary methods of solving them like on-policy reinforcement learning require a large number of interactions with the environment for every step of the optimization process. This can be prohibitively expensive in the real world, where the cost of each interaction can be high.

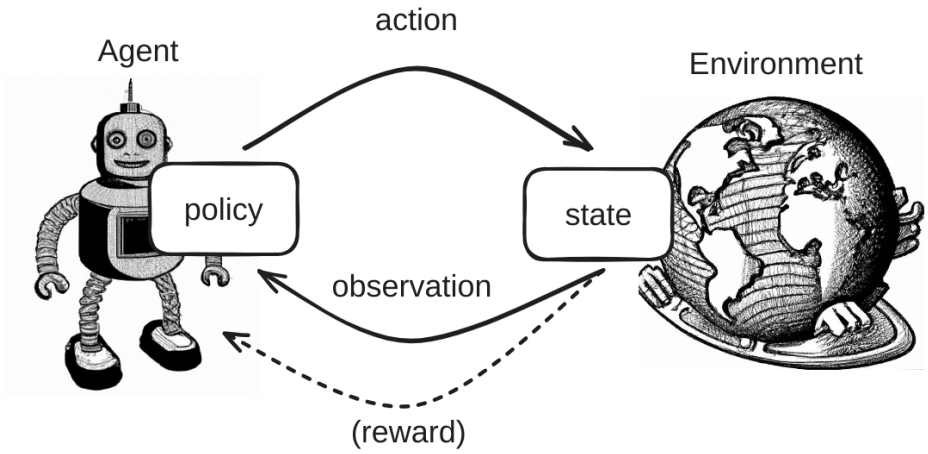


Figure 1.1: **Elements of a sequential decision making problem.** The agent interacts with the environment, and through its actions, it can alter the environment's state. In turn, the agent receives an updated observation of the environment based on the new state, and potentially, a reward. Utilizing its policy, the agent then determines the next action to take based on this recent observation. The images of the robot and globe were generated using DALLE-2 [51].

These challenges underscore the imperative of my research: to develop and explore efficient and cost-effective solutions for sequential decision-making problems.

1.1. Sequential decision making

The thesis investigates efficient machine learning methods for *sequential decision-making* problems. These types of problems require determining a series of actions to be taken in an environment in order to achieve a specific goal [20]. Many domains, such as autonomous driving, robotics, and game playing, can naturally be formulated as sequential decision-making problems.

However, the problem setup differs from the typical supervised learning setup, where each data point is assigned a ground truth label, and the datapoints are independent and identically distributed (i.i.d.). In sequential decision-making, the actions taken by the agent directly impact the state of the environment, which subsequently influences the agent's future observations. By taking different actions, the agent can explore various parts of the state space, thereby affecting the future distribution of the data it encounters. This contradicts the i.i.d. assumption, which is commonly made in supervised learning.

Sequential decision-making problems are typically represented by an agent engaging with its environment over multiple steps, as illustrated in Figure 1.1. The environment has a mutable state which changes over time. In a game of chess, for instance, this state corresponds to the positions of the pieces on the board and the progress of the game so far, which may impact the outcome. Alternatively, in a game of Texas Hold'em poker, the state includes the cards dealt to each player, those on the table, the remaining deck's order, each player's chip count, and the total money in the pot.

Each step in the decision-making process involves the agent observing the environment. This could mean direct access to the state, as in fully observable environments like chess, or access to only a portion of the state in partially observable environments, such as poker where

the agent does not know the cards held by the opponents or the order of the remaining deck. Based on these observations, the agent selects an action according to its policy – a function that can be stochastic and maps observations to actions. The chosen action, when applied to the environment, can alter its state.

The representation of a sequential decision making problem may also include a reward received by the agent from the environment. We are then faced with a reinforcement learning problem and learning goal is to design a policy that optimizes the sum of these rewards, which may be discounted over time.

Formalism. The sequential decision making problems are typically formalized as Partially-Observable Markov Decision Processes (POMDPs) [31]. They are characterized by a tuple $\langle \mathcal{S}, \Omega, \mathcal{A}, T, O \rangle$, where:

- $\mathcal{S}$, Ω, $\mathcal{A}$ are the space of all possible states, observations, and actions respectively.
- $T : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is a *state-transition function*, which maps a state and an action to a probability distribution over the next state.
- $O : \mathcal{S} \to \Pi(\Omega)$ is an *observation function*, which maps a state to a probability distribution over the observations. For fully observable environments, we can omit it, or assume $\Omega = \mathcal{S}$ and $O = id$.

For reinforcement learning the tuple is extended with:

- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward function*, which maps a state and an action to a real-valued reward.
- γ a discount factor. In reinforcement learning the goal is to optimize a discounted sum of future rewards: $\sum_{i=0}^{T} \gamma^i r_i$. If the future rewards should not be discounted, then $\gamma = 1$.

1.2. Machine learning approaches for sequential decision making

The sequential decision making problems can be solved using a variety of machine learning methods. A detailed review of these methods is beyond the scope of this thesis, but we will briefly discuss the most important ones.

1.2.1. Imitation learning

When the data used for training does not contain the reward signal, we encounter the challenge of *imitation learning* [29]. The objective is to learn a policy that mimics expert behavior. A prevalent method to address this is through *behavioral cloning*, which reframes the issue as a supervised learning task of predicting action based on observation. Another perspective on imitation learning is given by *inverse reinforcement learning* [4], an approach that strives to extract the reward function from the expert demonstrations, which could subsequently be used to train a policy. Additionally, there are sophisticated imitation learning strategies like *generative adversarial imitation learning (GAIL)* [25]. Drawing parallels with generative adversarial networks (GANs) [21], GAIL tries to train a policy that will be indistinguishable from the expert's policy by an adversarial discriminator.

We present a set of methods using imitation learning for autonomous driving in Chapter 4. In Chapter 5, the low level controller incorporated from ViNG [56] is trained by supervised learning, and can be interpreted as a variant of behavioral cloning.

1.2.2. Reinforcement learning.

When the reward signal is available, we can use reinforcement learning methods to train a policy. Reinforcement learning algorithms can be classified in various ways. One classification hinges on the type of data used during the training process. Here, we distinguish between *on-policy*, *off-policy*, and *off-line* or *batch* [37]. The on-policy methods utilize data solely from the current policy. In contrast, the off-policy approaches can incorporate additional data, while the off-line methods exclusively use pre-recorded data, with no possibility of interacting with the environment.

Another categorization of reinforcement learning algorithms is based on model usage, segregating them into *model-free* and *model-based*. Model-based methods use an explicit model of the environment, while model-free methods do not. For a deeper discussion of model-based methods, refer to Chapter 2.

Furthermore, these algorithms can be differentiated by the type of policy used: *value-based*, *policy-based*, and *actor-critic*. Value-based policies focus on estimating the quality of each state or state-action pair. Policy-based methods, on the other hand, train a policy directly, while actor-critic methods combine the two approaches.

Chapter 3 demonstrates usage of Proximal Policy Optimization (PPO) [55] which is an example of a modern, on-policy, actor-critic method.

1.3. Reducing data acquisition costs

As argued at the beginning of this chapter, the cost of collecting data for sequential decision making problems, especially those positioned in the real world, is one of the main obstacles to the widespread adoption of machine learning methods in this field. This thesis aims to address this issue by presenting results from three high-level strategies that can help reduce data acquisition costs. These strategies involve making algorithms more sample efficient, utilizing cheaper data, and leveraging knowledge encoded in pre-trained machine learning models. In the following sections, we will elaborate on each of these strategies and reference chapters where they are discussed in detail.

To enhance sample efficiency, we propose employing model-based reinforcement learning. In Chapter 2, we present the results of applying these methods to the visual domain – Atari games.

In the line of work on reducing data acquisition costs, we explore methods applicable to autonomous driving. Chapter 3 demonstrates the use of simulated data to train a driving policy that can be deployed in the real world. Additionally, Chapter 4 summarizes a set of approaches that enable the deployment of a policy trained with imitation learning. Collecting data for imitation learning is less expensive and safer as it only requires an expert driver to demonstrate the desired behavior, eliminating the risks associated with trial and error processes in real-world reinforcement learning.

Lastly, we investigate the utilization of knowledge embedded in pre-trained models. Recent progress in large language models [32], exemplified by the introduction of ChatGPT [45], has showcased the potential of scaling up models trained with self-supervision over extensive data and applying them to a variety of downstream tasks. In Chapter 5, we illustrate how pre-trained models for language, vision, and control can be applied to robotics. The resulting system, named LM-Nav, is capable of following natural language instructions in the real world without the need for training or fine-tuning.

Chapter 2

Model-based approach in visual domains

Model-based reinforcement learning (MBRL) is a powerful approach to solving sequential decision-making problems by leveraging an explicit model of the environment. This class of methods can either incorporate a pre-specified model (e.g., in board games such as chess, with the rules provided upfront) or learn the model from available data. While MBRL can be more challenging to train compared to model-free methods, it tends to be superior in terms of sample efficiency, which is the main motivation for pursuing it (refer to Chapter 7.1 in [42]).

In real-world applications, reinforcement learning shows great promise when integrated with visual inputs, such as camera-equipped robots or autonomous vehicles. However, the application of model-based methods in visual domains encounters a substantial challenge due to the high dimensionality of the observation space. The number of data features of image-based observation spaces is many orders of magnitude greater than domains where earlier iterations of MBRL, like PILCO [15], were successful.

A particularly important benchmark for assessing sequential decision-making in visual domains is the Arcade Learning Environment (ALE) [6]. Since the inception of deep reinforcement learning algorithms, starting with the pioneering DQN [41], this environment has become the standard benchmark for reinforcement learning techniques. However, until recently, there had been *"no demonstration of successful planning with a learned model in the ALE"* (ref Section 7.2 in Machado et al. [39]).

This chapter contains an overview of our pioneering work on applying MBRL to Atari games. It was the first to show that introducing a video prediction model to a deep reinforcement learning algorithm outperforms the then state-of-the-art model-free algorithms in sample efficiency, even by over an order of magnitude in some games. The impact of the paper can be measured by its citations - over 700 at the moment, according to Google Scholar. The sections below delve into the specific aspects of the work.

2.1. Model-Based Reinforcement Learning for Atari

We proposed an algorithm called Simulated Policy Learning (SimPLe). The method follows an iterative process that involves the following steps:

1. Collect data from the environment using the latest policy. Initially the policy is random.
2. Train a world model on the collected data. The world model is an action conditioned video prediction model, which generates the next likely frame and reward given the

current frame and action.

3. Train a policy inside the world model. The policy is being trained using a standard on-policy algorithm: Proximal Policy Optimization (PPO) [55].

By using the world model to train the policy, the overall amount of data from real Atari games is significantly reduced. We demonstrated that SimPLe can learn to play multiple Atari games using only 100K interactions with the real game, which is about two hours of gameplay. This is a significant improvement over the previous model-free methods, which typically required millions of interactions with the environment.

2.1.1. World-model architecture

The world model architecture is similar to that of an autoencoder, with convolutional layers squeezing 4 input frames into the bottleneck layer, and then deconvolutional layers reconstructing the next frame. In order to assure that the action is not ignored, it is attached to every layer of the decoder. The key novelty comes from adding a discrete latent variable to the bottleneck layer. The idea is similar to that of a variational autoencoder (VAE) [35] and the discrete form of the latent variable was selected due to *discrete* nature of the Atari games.

(a) Freeway (b) Kung-Fu Master

Figure 2.1: **Examples of world-model of Atari.** In each of the images, the left panel is the model output, the middle panel is the ground truth, and the right panel is the absolute difference between the two. In the case of Freeway, the prediction is *pixel-perfect*, hence the black panel. In the case of Kung-Fu Master, the world model predicts a different number of opponents, but other than this the prediction is plausible. Please note that the action of the player (high kick) is correctly captured.

Even though the world model is designed to predict only a single frame, it can be deployed autoregressively to generate a sequence of frames of any desired length. Our observations indicate that our model, in numerous games, is capable of generating extended sequences that either precisely match the ground truth or produce a distinct yet plausible series of frames. Examples of this can be seen in Figure 2.1.

2.1.2. Policy training

The policy is trained on a small set of experiences collected during model exploration (step 1), and a large number of rollouts that are generated from the world model using PPO. However, a significant challenge comes from the fact that even the best world model tends to produce low-quality data when unrolled autoregressively for an extended number of steps. This poses a problem when training agents to play Atari games, where a single episode can involve thousands of steps.

Our solution to the problem comes in the form of *random starts*. Instead of unrolling the entire trajectory in the world model, we select random starting points from the trajectories collected in step 1 and autoregressively unroll from the world model for a limited number of steps (50 or 100 in practice).

2.1.3. Evaluation and Atari 100k

For the evaluation of our method, we used a fixed budget of 100K interactions for each of the games. This was a significant reduction compared to the previous evaluation methods, which typically used 10M interactions [41, 55].

The evaluation of reinforcement learning in such a low data regime has since become a new benchmark for algorithms, commonly referred to as *Atari 100k*. This benchmark has since been adopted by many significant studies, such as the one by Micheli et al. [40] which advocates for the use of transformers as world models. An even more noteworthy example is the paper by Agarwal et al. [1], which received an Outstanding Paper Award at NeurIPS 2021. This work illustrates how performance indicators can become unreliable when too few training runs are reported. It is encouraging to observe that in this independent evaluation, SimPLe has proven to be competitive when compared to some of the more recently proposed methods including CURL [36].

2.2. Summary

This chapter contains a discussion on the subject of model-based reinforcement learning. An application of this concept is introduced via SimPLe, a model-based approach designed for Atari games. This method achieved state-of-the-art results in low-data regime. Furthermore, the work presented in this chapter has spurred interest in evaluating agents trained within a fixed budget of 100K interactions, a criterion now widely recognized as the Atari 100k benchmark.

Chapter 3

Sim-to-real policy transfer for autonomous driving

This chapter focuses on autonomous driving, the central theme that spans both this chapter and the following one. We begin by providing an overview of the field, exploring its challenges and potential solutions. Subsequently, we explore the utilization of simulators as an alternative for data collection. Lastly, we introduce our novel methodologies for transferring driving policies from the simulated environment to real-world scenarios, thereby bridging the gap between virtual and physical domains.

3.1. Autonomous driving

Enabling autonomous driving presents a significant interdisciplinary challenge, encompassing robotics, computer vision, sequential decision making, and machine learning. Successfully addressing this challenge in a scalable manner holds immense potential for societal impact, including enhanced safety, reduced transportation costs, and increased accessibility. Notably, while recent years have witnessed substantial progress in the field, with a few notable real-world deployments [59], it is important to recognize that autonomous driving research has a much longer history than commonly perceived.

As early as the 1980s, pioneering projects such as ALVINN [48] and VaMoRs [18] demonstrated the ability to drive on highways. However, despite four decades of development and significant advancements, numerous challenges persist. The existing approaches do not effectively scale to a wide range of locations, as evidenced by the limited active deployments in just a couple of cities, despite substantial funding in the billions of dollars. This indicates that autonomous driving remains a far from solved problem [24, 30].

To understand the reason for these limitations we need to take a closer look at a typical autonomous driving system. It consists of multiple components, including perception (e.g. object detection, semantic segmentation), prediction (e.g. motion forecasting for other traffic participants), and planning (e.g. trajectory planning), see Figure 3.1. According to Jain et al. [30] and Hawke et al. [24], unlike perception and in growing manner prediction, planning still mostly relies on complex hand-crafted rules. This, in turn, negatively impacts the cost and speed of the development of self-driving systems. The lack of technical scalability calls for a paradigm shift towards data-driven solutions utilizing machine learning, to achieve greater scalability. This and the next chapter aim to explore methods that represent steps in this direction.

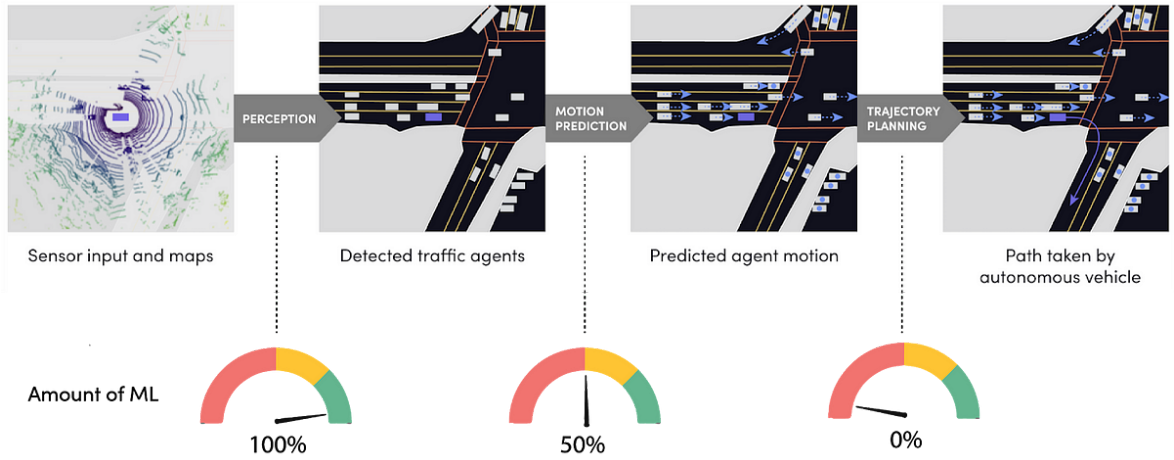


Figure 3.1: **A modern autonomous driving pipeline with the amount of machine learning used in each component**. While perception and predictions heavily use machine learning, planning still mostly relies on hand-crafted rules. Figure adapted with author's permission from: [44].

3.2. Simulator as an alternative to real-world data collection

Simulation has a number of advantages over real-world data collection. First, it is typically much cheaper to generate data, and the process is not wearing out the equipment (be it robot manipulator, or autonomous vehicle). Secondly, it is possible to generate a vast amount of data in a short time, thanks to the speedup and parallelization of the simulation. Finally, simulators can generate data from situations that are dangerous without any risk to the equipment or the operator.

However, there are crucial differences between the real world and the simulation, sometimes referred to as the *sim-to-real gap*. See Figure 3.2 for an example of such a gap in the context of camera input in autonomous driving. A policy trained purely in a simulator is very unlikely to perform well in the real world. In this chapter, we will discuss the results of our methods to tackle the sim-to-real gap in the context of autonomous driving. We will start by shortly describing the problem space, which is also shared with the next chapter.

(a) Simulation (b) Reality

Figure 3.2: **Sim-to-real gap in autonomous driving.**

3.3. Simulation-based reinforcement learning for real-world autonomous driving

In our study, we present evidence of the successful transfer of a driving policy that was initially trained in the Carla simulator [19] to a real-world vehicle. This accomplishment required a comprehensive exploration of various approaches. We trained and evaluated ten distinct classes of models, meticulously assessing their performance in both simulated and real-world environments. To ensure the robustness of our findings, we conducted approximately 400 test runs in the real world.

It is important to note that this research was conducted in collaboration with our industry partner, Volkswagen, adding real-world practicality and validation to our work.

Domain radnomization. The key enabler of sim-to-real transfer was a technique known as *domain randomization* [60]. To develop a policy that can withstand the sim-to-real gap, it is crucial to expose the policy to a broad spectrum of situations within the simulation. We incorporated extensive perturbations into the simulator, such as alterations in lighting, weather, and textures, as well as Gaussian noise and cutouts [17] of the camera image. As expected, we observed improved transfer between different domains.

Regularization. During our investigation, we made intriguing observations that are often considered intuitive in other areas of machine learning but were less emphasized in the context of reinforcement learning. For instance, we discovered that the regularization of neural networks plays a crucial role in facilitating the successful transfer of policies from simulation to reality.

Offline evaluation metric. Additionally, we recognized the pressing need for a metric that could offer valuable insights into the performance of the learned policy before its deployment in the real world. To address this, we proposed a novel metric that proved to be highly beneficial in providing a preliminary assessment of real-world performance. It was a mean absolute error between the steering angles of a human driver and the learned policy on a recorded trajectory.

3.4. Summary

This chapter offers a succinct introduction to the realm of autonomous driving, highlighting the considerable potential of simulators as substitutes for real-world data collection. It further elaborates on research conducted in collaboration with Volkswagen, focused on the transfer of driving policies from simulation to reality. Significant progress has been accomplished in improving the sim-to-real transition of driving policies, largely due to the application of domain randomization and regularization. Additionally, the introduction of an offline evaluation metric presents a potential to increase the velocity of the development of similar approaches in the future.

Chapter 4

Imitation learning for autonomous driving

Imitation learning is a well-established approach for solving sequential decision-making problems using supervised learning. Its core principle is to learn a policy that mimics the behavior of an expert, which is demonstrated through a series of actions taken in a given environment. Applying this approach to autonomous driving is an old idea, in fact, this was the technique used by ALVINN [48] in 1990. The interest in these type of approaches was rekindled by the method called ChauffeurNet [5] in 2019. It combined advanced perception with imitation learning to create a policy capable of driving on a realistic test track.

Imitation learning offers a cost-efficient approach for autonomous driving due to its relative ease in collecting a large dataset of expert demonstrations. The process simply involves allowing a human driver to operate a sensor-equipped car, generating a valuable dataset for training purposes. This is in contrast to on-policy reinforcement learning, which requires a large number of interactions with the environment to learn a policy.

For these reasons, the research team at Lyft Level 5 (later transferred to Woven Planet, a subsidiary of Toyota) embarked on an exploration of imitation learning as a promising solution for real-world autonomous driving. I had the privilege of working with this team from 2020 to 2022. The team achieved significant advancements in this endeavor, resulting in the successful transition of the company's prototype stack from a heuristic-based policy to an imitation learning-based policy. Some of our notable scientific contributions have been published and are summarized in this chapter.

4.1. What data do we need for training an AV motion planner?

Training a machine learning planner requires a vast dataset of expert demonstrations. Houston et al. [27] has demonstrated that the performance is steadily improving until at least 1000 hours of data are used for training. Further advancements likely call for an even larger dataset, which underscores the importance of efficient data collection.

Autonomous vehicles are usually fitted with a range of sensors, including cameras, lidars, radars, and GPS. These tools make them capable of autonomous driving, but they also contribute to high equipment and operational costs. In this study, we question whether a low-fidelity dataset collected using less expensive sensor suites (e.g., cameras only) can effectively be used to train a performant planner.

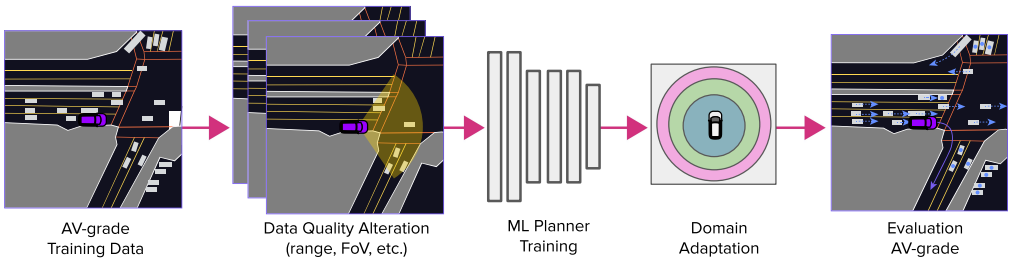


Figure 4.1: **The methodology of comparing the impact of data quality on an ML planner performance.**. Figure adapted with permission from [14].

To answer the question, we adopted a methodology as outlined in Figure 4.1. We started with a large, high-quality dataset collected using a full sensor suite. This data was then deliberately downgraded in quality, mimicking the outcome of using less expensive sensors. We then compared the performance of a planner trained on datasets of varying size and quality. The performance of planners trained on datasets of various sizes and qualities was then compared.

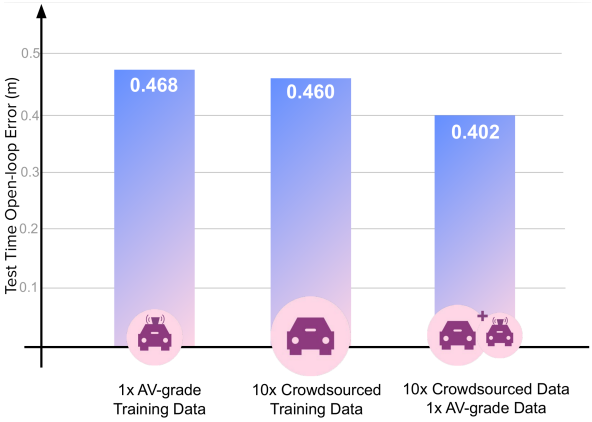


Figure 4.2: **Data quality comparisson**. Y-axis represents open-loop error, lower is better. Figure adapted with permission from [14].

Our findings, summarized in Figure 4.2, suggest that a larger dataset of lower-quality data may, in fact, be more beneficial than a smaller dataset of higher-quality data. The most effective approach appears to be pre-training the model on low-quality data and then fine-tuning it with high-quality data.

These results significantly impact the economics of data collection. This has particular importance for companies with large vehicle fleets, such as Lyft. The feasibility of installing less expensive sensor suites in numerous vehicles to gather the necessary data for training a high-quality planner offers an economically viable solution.

4.2. SimNet: Learning reactive self-driving simulations from real-world observations

While developing a proficient autonomous driving planner is a critical aspect, having methods to accurately verify its performance is equally essential. We need to carry out tests within a safe environment where potential mistakes will not have costly and dangerous consequences. Naturally, simulators provide an ideal setting for such evaluations. However, they are not without their limitations, particularly concerning the realistic representation of other traffic participants' behavior. Typically, behaviors are simulated either through rule-based heuristics or so called *log-replay*, which involves replaying recorded behavior [13]. The latter approach is widely used, but it has a critical issue. When the evaluated agent, also known as *ego vehicle*, takes a different path than the recorded one, the behavior of other participants remains unchanged. This results in simulations that do not accurately reflect reality, which can be misleading and cause both false positive and false negative errors. An example of a false

positive might be a scenario where the ego vehicle is driving slower and ends up being rear-ended. An instance of a false negative will be discussed below. In response to this challenge, we propose a method that learns the behavior of other traffic participants directly from real-world data, thus offering a more authentic interaction model.

In the study, we introduce a method that learns the behavior of other traffic participants directly from real-world dataset collected by a fleet of autonomous vehicles (see Houston et al. [27] for the dataset details). This approach, which we named SimNet, aims at providing a more realistic model for traffic interaction than log-replay and scripted heuristics. We evaluate the effectiveness of this method based on its realism (measured by the distance to actual trajectories when the ego vehicle follows a recorded path) and reactiveness (how simulated agents respond to the ego vehicle stopping). We observed promising results in both aspects. Furthermore, the performance improves as more data is incorporated, indicating the potential for further scalability. This shows the promise for future enhancements that can be achieved not through substantial engineering efforts, but simply by adding more data to the system.

As mentioned before, SimNet helped us to discover an interesting example of our machine learning-based planner that remained hidden with log-replay. When we compared the performance of the planner evaluated with these two methods, we observed significant differences. In particular, when evaluated with SimNet, the planner was failing to start at a green traffic light, a behavior that was not present with non-reactive agents. Upon investigation, we determined that the ML planner succumbed to what is known as *causal confusion*. It had learned to start moving not when the light turned green, but rather when the car behind it began to move. Such a strategy works perfectly well with log-replay. However, when the agents are reactive (as in SimNet, or in the real world), the vehicle behind does not start moving until the ego vehicle does, resulting in a deadlock. This example highlights the significance of realistic simulation, and that SimNet is a promising step in this direction.

4.3. Urban driver: Learning to drive from real-world demonstrations using policy gradients

Training a control policy using imitation learning poses certain challenges. A common issue arises from covariate shift, a situation where the policy deviates slightly from the expert, leading it to encounter data distributions not seen during training. On the unseen data, it is more likely to make mistakes, the errors will accumulate, and drift further away from the expert. One algorithm proposed to tackle this issue is DAgger [53]. It iteratively trains a policy, applies it in the target environment, and then invites an expert to correct the policy behavior. While DAgger provides theoretical performance guarantees, the requirement to query an expert makes it hard to scale. In the context of real-world autonomous driving, it could potentially be dangerous.

CheuffeurNet [5] proposes an alternative method to address the covariate shift problem. It incorporates perturbations into the recorded trajectories during training. Instead of solely using the original trajectory, it slightly displaces the vehicle to the left or right, subsequently using the altered trajectory as a label. However, this seemingly scalable approach brings its own set of issues. The perturbations can be hard to calibrate, and in reality, perturbations are needed on several more factors, such as speed and the behaviors of other vehicles. This renders the method quite complex to fine-tune and limits its scalability, contrary to initial expectations.

Drawing inspiration from these methods, we introduced Urban Driver, a novel algorithm that attempts to address the covariate shift problem without the need to query an expert or

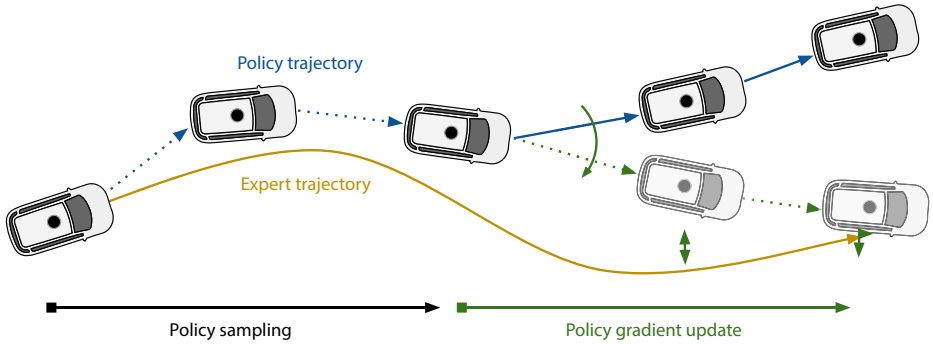


Figure 4.3: **Training procedure for Urban Driver**. The policy is unrolled in a simulator for a number of steps. The first K is used only for sampling. The rest is used to update the policy using policy gradient with the loss being the distance to the expert trajectory. Figure adapted with permission from [54].

artificially perturb the data. It operates similarly to a reinforcement algorithm, by collecting trajectories through unrolling current policy in a simulator, see Figure 4.3. Instead of a reward, it uses the distance to the ground truth (recorded) trajectory as a loss function for each step after the initial K steps. These initial steps are used to make sure that sampling is done outside of the expert distribution. Taking advantage of the simulator's differentiability, it conducts backpropagation through time (BPTT), enabling more nuanced adjustments to the trajectory over time. In particular, we observed that using BPTT allows for easy tweaking of the model performance, e.g. trading off between safety and comfort.

A similar technique was previously introduced by Venkatraman et al. [62] in the context of multi-step predictions. However, our method is distinctive in the usage of BPTT and application to a control problem.

4.4. SafetyNet: Safe planning for real-world self-driving vehicles using machine-learned policies

Machine Learning (ML) planners offer numerous advantages, the most critical of which is their capacity to scale performance based on data rather than engineering effort. This quality allows them to solve problems that would traditionally be intractable for classical rule-based heuristics. However, a significant challenge with these ML planners is the difficulty in assuring their safety. To address this concern, we propose the integration of a rule-based system as a fallback layer for the ML planner. We have developed a comprehensive system, which we call SafetyNet, that incorporates this idea. Demonstrating its effectiveness in a real-world application, SafetyNet has been successfully implemented to control a self-driving vehicle navigating the streets of San Francisco, see bottom images in Figure 4.4.

The operation of the SafetyNet system initiates with the generation of a candidate trajectory using an ML planner. Subsequently, this proposed trajectory is passed through a rule-based trajectory evaluator that verifies its safety and legality. If the evaluator deems the trajectory as appropriate, the system proceeds to execute it. However, if the trajectory fails to meet the established criteria, the rule-based fallback layer is triggered. This component then takes over the responsibility of generating a new, compliant trajectory.

The interplay between both components of the SafetyNet system can be best illustrated with a simple example of a self-driving vehicle following a lead vehicle. Even though the task seems simple, presents significant complexities in developing a rule-based system that can

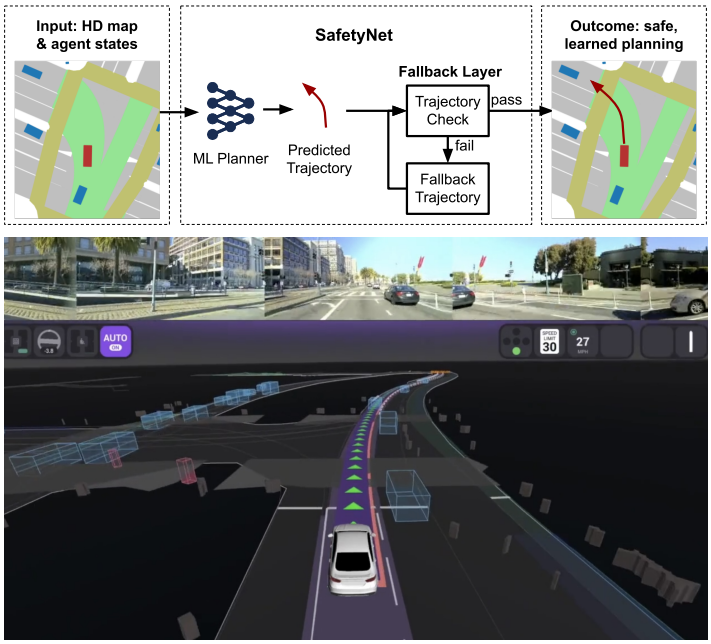


Figure 4.4: *Top:* SafetyNet combines ML planner with a rule-based layer. *Bottom:* SafetyNet deployed on the streets of San Francisco. Figure adapted with permission from [64].

handle all possible scenarios, and also take into account parameters such as riders' comfort. The system must manage numerous parameters, such as determining the preferred distance from the lead vehicle — which must vary based on speed — along with preferred acceleration and deceleration rates, etc. The ML planner is designed to implicitly learn these preferences from the data provided, consistently producing suitable trajectories most of the time. However, due to the nature of probabilistic models, they can occasionally produce a trajectory that is not safe.

That is why the ML planner's decisions are then vetted by the rule-based fallback layer to ensure the suggested trajectory is indeed safe. It validates that the trajectory avoids collisions with other vehicles, does not exceed speed limits, and prevents the vehicle from hitting the curb. These checks are comparatively straightforward to implement within a rule-based system. By utilizing these two approaches in parallel, we can develop a system that is not only safe but also highly performant, effectively marrying the adaptability of a machine learning planner with the strict safety regulations of a rule-based system.

4.5. Summary

This chapter explores a series of enhancements aimed at boosting the efficiency of an autonomous driving system with a machine learning-based planner. The findings suggest that this planner can be effectively trained using a large amount of low-fidelity data, which is a more economically viable option for data gathering. The creation of a data-driven simulator has also been discussed, which aids in validating and exposing the nuances of a planner without the high costs linked to real-world deployment. Additionally, a novel training algorithm called Urban Driver is introduced, which eliminates the necessity for querying experts or introducing artificial data perturbations during planner training. The proposal of SafetyNet, a system that enhances planner safety by integration a rule-based system, is also discussed. Collectively, these advancements have facilitated successful system deployments on the bustling streets of San Francisco.

Chapter 5

Foundation models for decision making

Foundation models is a recently coined term referring to models that are trained on extensive and diverse datasets, making them highly applicable across a wide range of downstream tasks [7]. This paradigm shift was initiated by influential large language models such as BERT [16] and GPT-3 [9]. These models are characterized by their substantial number of parameters, with GPT-3 alone boasting an impressive 175 billion parameters. Furthermore, they are trained on exceptionally large datasets, exemplified by Chinchilla's utilization of 1.4 trillion tokens [26]. Apart from pure language models, the foundation model paradigm has also been applied to multimodal tasks, such as vision and language [3, 49]. The significant scale and versatility of these foundation models have opened up many new research directions and even claims about their potential to achieve general artificial intelligence [11].

The extensive knowledge embedded within foundation models holds great potential for systems capable of interaction with the real world, such as robotics. In recent years, there has been a growing interest in pursuing this avenue of research. The first area of application is to improve the task specific performance, by relying on large, pre-trained models. Notable examples include Radosavovic et al. [50], where a large pre-trained vision model pre-trained on self-supervised tasks is successfully applied to robotic manipulation. Moreover, foundation models can also enable new capabilities, such as providing a natural language interface to a robot. An impressive demonstration of this approach can be found in Brohan et al. [8], where a combination of simple reasoning using a large language model and robotic affordances leads to a robot completing vague instructions expressed in natural language, such as the request *"I spilled my drink, can you help?"*.

In this chapter, we present a work that leverages the power of large, pre-trained models to enable robotic navigation.

5.1. LM-Nav: Robotic navigation with large pre-trained models of language, vision, and action

In this work, we tackle the challenge of navigation based on natural language instructions. Here is an illustrative example of such an instruction: *"Take a right next to a stop sign. Look for a glass building, after passing by a white car"*. It is evident that to execute these instructions accurately, the robot must possess a comprehensive understanding of the world, encompassing knowledge of objects like stop signs and glass buildings. Previously, the prevailing approach involved creating a dataset consisting of instructions paired with corresponding

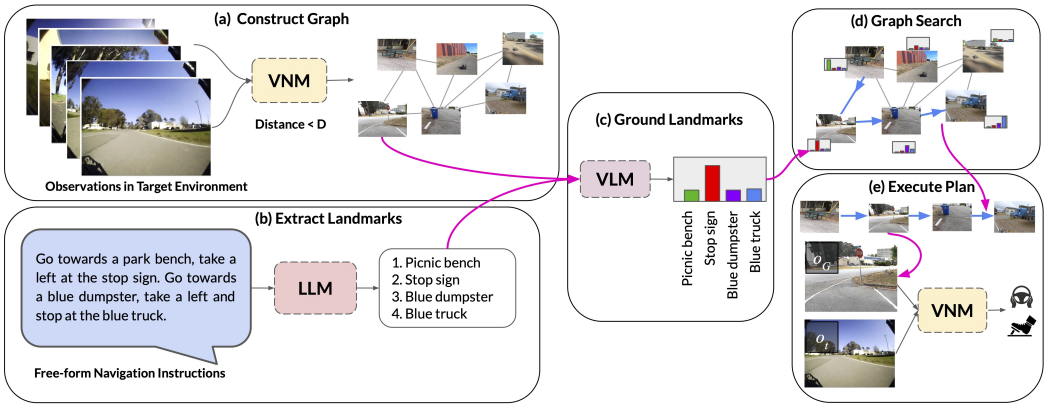


Figure 5.1: **System overview of LM-Nav**. Figure adapted with permission from [57].

trajectories, which was then utilized to train a navigation model. A recent example of such work is Vasudevan et al. [61]. However, in our pursuit of more efficient methods, we propose harnessing the capabilities of pre-trained models to solve this task.

Our approach seeks to deconstruct the task of navigation into distinct sub-tasks, each addressed by a specific pre-trained model or an optimization algorithm. The system overview is depicted in Figure 5.1. We operate under the assumption that the robot is equipped with knowledge about its environment and possesses access to a graph outlining its traversable paths.

The navigation process begins with an instruction that needs to be accurately interpreted. For this task, we employ the large language model (LLM), GPT-3 [10], to extract the relevant landmarks from the instructions provided. Subsequently, the second sub-task involves grounding these extracted landmarks within the graph of the environment. In order to associate each pair of landmarks and graph vertices, we employ the vision-and-language model (VLM), CLIP [49]. This model estimates the likelihood of the presence of a particular landmark at each vertex. The subsequent step in the process involves the planning of an optimal trajectory. For this, we developed a custom dynamic programming algorithm that finds a path that both maximizes the likelihood of visiting all extracted landmarks and minimizes the distance traversed. Lastly, the execution of the planned trajectory forms the final sub-task. This is handled by a self-supervised vision navigation model (VNM), ViNG [56], which brings the plan into actual motion.

The system is called LM-Nav, standing for Language Model Navigation. Its distinctive attribute is that it does not require any training or even fine-tuning of the incorporated models. This allows LM-Nav to be regarded as the ultimate sample efficient method, as it completely negates the necessity for data collection for training purposes.

5.2. Summary

This chapter outlines the progress made in the development and applications of large, pre-trained models, often referred to as foundation models. The potential of these models to guide a robot based on natural language instructions is demonstrated through the system called LM-Nav. Notably, LM-Nav achieves impressive real-world performance without relying on any data for training or fine-tuning the models. As such, it can be regarded as the pinnacle of sample-efficient methods for navigation.

Chapter 6

Conclusions and future directions

This dissertation encapsulates approximately six years of extensive work on efficient methods for sequential decision-making across diverse applications such as Atari games, autonomous driving, and robotics. It explores a variety of methods, including model-based reinforcement learning (RL), sim-to-real transfer, imitation learning, and the utilization of foundation models. In the course of this research, several pioneering achievements were accomplished. These include the first-ever successful application of model-based RL to Atari games, and the first-ever real-world deployment of a full-sized car that was trained in a simulated environment using RL.

Interestingly, despite the apparent diversity of these approaches, they all hint towards a common direction. Future research is expected to witness a greater convergence and simultaneous application of these methods. Early indications of this trend are already traceable in recent literature. For instance, recent work by Hu et al. [28], proposes the integration of model-based and imitation learning for autonomous driving. Simultaneously, with the tremendous advancement in foundation models, their application to robotics and RL is anticipated to surge. Alex Kendall, CEO of autonomous driving startup Wayve, recently discussed the potential of deploying large language models for autonomous driving, further solidifying this prediction [34].

In my opinion, the path to scalable autonomous systems in the real world will necessitate pretraining large models on vast amounts of relevant, albeit imperfect, data, followed by fine-tuning these models on the targeted task. It might be even possible to conduct the fine-tuning just before the task executions, similarly to how large language models are prompted for a specific task. The growing availability of large scale dataset, such as ego4d [23] or "something something" [22], will enable this approach. Intriguingly, this recipe is similar to the strategy discovered in our work on utilizing low-quality data for autonomous driving, as discussed in Section 4.1. Similarly, the recent study by Ouyang et al. [47] exemplifies, refining GPT-3 with human feedback facilitated the creation of a model, known as InstructGPT. Even though this model has 100 times fewer parameters, it surpasses the performance of the much larger GPT-3 model in aligning more closely with user intent. Given the rapid advancements in machine learning and the increasing ubiquity of data, this blend of large-scale pretraining and fine-tuning holds immense promise for the development of autonomous systems, paving the way for more robust, adaptable, and efficient real-world applications.

Chapter 7

Acknowledgements

I would like to thank my supervisor, Piotr Miłoś, for his guidance, knowledge, companionship, and support. I would not be here also without the help from Henryk Michalewski, who was supervising me during the early stages of my PhD.

I was fortunate to spend a few months as a Fulbright Scholar at the University of California, Berkeley, where I was hosted by Sergey Levine. Prior to this I was at Google Brain in Mountain View, where I was hosted by George Tucker and Łukasz Kaiser. I learned a lot from these experiences, and I am grateful for the opportunity to work with such amazing people.

I also wanted to thank Peter Ondruska, who was my manager at Lyft Level 5 and Woven Planet. I am grateful for the opportunity to work with him and learn from him, and for pushing the envelope of *data-driven driving*.

I also would like to acknowledge Robert Bogucki, who was my supervisor at deepsense.ai. The decision to start a research team working on Atari at a machine learning consultancy company was very bold, and I am grateful for the opportunity to be a part of it.

I would also like to express my gratitude towards all remaining collaborators: Adam Jakubowski, Afroz Mohiuddin, Ana Ferreira, Antonia Breuer, Ashesh Jain, Brian Ichter, Chelsea Finn, Chih Hu, Christopher Galias, Damian Stachura, Dhruv Shah, Dumitru Erhan, Hugo Grimmett, Konrad Budek, Konrad Czechowski, Long Chen, Luca Bergamini, Luca Del Pero, Lukas Platinsky, Łukasz Kidziński, Maciej Wołczyk, Matt Vitelli, Michael Equi, Michał Martyniak, Michał Zawalski, Mohammad Babaeizadeh, Moritz Niendorf, Oliver Scheel, Paweł Zięcina, Peter Ondruska, Piotr Kozakowski, Piotr Tarasiewicz, Qiangui Huang, Ryan Sepassi, Sergey Levine, Silviu Homoceanu, Stefanie Speichert, Yan Chang, Yawei Ye.

Last, but not least, I would like to thank my family and friends. First, and foremost, to my wife Albertyna, for her love and believing in me more than I believe in myself. My mother, Aniela Osińska, for her love and support, and always encouraging me to go further... and faster. My late father, Jan Osiński, for his support and being a source of the most surprising stories. My daughter, Wanda Osińska, for being a source of joy and inspiration. My brother, Jędrzej Osiński, for his support and always challenging me. My in-laws, Ewa Zalewska-Paciorek and Włodzimierz Paciorek, and my broher-in-law, Adam Paciorek, for their support and warmly welcoming me to the family. My friends for always being there for me over the years: Wojciech Śmietanka, Artur Żyliński, Piotr Gałka, Marcin Wojno, Mateusz Zarudzki, Marta Jedraska, Tomasz Mańka, Aneta Wiśniewska, Michał Lemańczyk, Blanka Domachowska, Aleksandra and Jan Szumieł, Karolina and Maciej Żak, Grażyna Chlebicka, Filip Kiałka, Marcin Moczulski.

Bibliography

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

[2] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[3] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

[4] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[5] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26*, 2019.

[6] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[7] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, and Emma Brunskill et al. On the opportunities and risks of foundation models. *ArXiv*, 2021. URL `https://crfm.stanford.edu/assets/report.pdf`.

[8] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318. PMLR, 2023.

[9] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365 (6456):885–890, 2019.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[11] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[12] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[13] Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15590–15599, 2021.

[14] Long Chen, Lukas Platinsky, Stefanie Speichert, Błażej Osiński, Oliver Scheel, Yawei Ye, Hugo Grimmett, Luca Del Pero, and Peter Ondruska. What data do we need for training an AV motion planner? In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1066–1072. IEEE, 2021.

[15] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[17] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[18] Ernst Dieter Dickmanns and Volker Graefe. Applications of dynamic monocular machine vision. *Machine vision and Applications*, 1(4):241–261, 1988.

[19] 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.

[20] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[22] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The" something something" video database for learning and evaluating visual common sense. In *Proceedings of the IEEE international conference on computer vision*, pages 5842–5850, 2017.

[23] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012, 2022.

[24] Jeffrey Hawke, Vijay Badrinarayanan, Alex Kendall, et al. Reimagining an autonomous vehicle. *arXiv preprint arXiv:2108.05805*, 2021.

[25] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[26] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[27] John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Long Chen, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset. In *Conference on Robot Learning*, pages 409–418. PMLR, 2021.

[28] Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zachary Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. Model-based imitation learning for urban driving. *Advances in Neural Information Processing Systems*, 35:20703–20716, 2022.

[29] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[30] Ashesh Jain, Luca Del Pero, Hugo Grimmett, and Peter Ondruska. Autonomy 2.0: Why is self-driving always 5 years away? *arXiv preprint arXiv:2107.08142*, 2021.

[31] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[32] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[33] Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. Gpt-4 passes the bar exam. *Available at SSRN 4389233*, 2023.

[34] Alex Kendall. 2023's ai breakthroughs apply to avs too, accelerating wayve's foundation model for embodied ai. *Waymo Blog*, 2023. URL `https://wayve.ai/thinking/2023_ai_breakthroughs_apply_to_avs_too/`.

[35] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[36] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR, 2020.

[37] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[38] Yinlin Li, Peng Wang, Rui Li, Mo Tao, Zhiyong Liu, and Hong Qiao. A survey of multifingered robotic manipulation: Biological results, structural evolvements, and learning methods. *Frontiers in Neurorobotics*, 16, 2022. doi: 10.3389/fnbot.2022.843267.

[39] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61: 523–562, 2018.

[40] Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample efficient world models. *arXiv preprint arXiv:2209.00588*, 2022.

[41] 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.

[42] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[43] Hans P Moravec. *Mind children*. Harvard University Press, 1988.

[44] Peter Ondruska and Sammy Omari. The next frontier in self-driving: Using machine learning to solve motion planning. *Medium*, 2021. URL `https://medium.com/lyftself-driving/the-next-frontier-in-self-driving-using-machine-learning-to-solve-motion-planning-a259b814e9ad`. Accessed: January 22, 2021.

[45] OpenAI. Introducing chatgpt. *OpenAI Blog*, November 2022. URL `https://openai.com/blog/chatgpt`.

[46] OpenAI. Gpt-4 technical report, 2023.

[47] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[48] Dean Pomerleau. Rapidly adapting artificial neural networks for autonomous navigation. In R.P. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3. Morgan-Kaufmann, 1990.

[49] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[50] Ilija Radosavovic, Tete Xiao, Stephen James, Pieter Abbeel, Jitendra Malik, and Trevor Darrell. Real-world robot learning with masked visual pre-training. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 416–426. PMLR, 14–18 Dec 2023.

[51] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[52] Guinness World Records. Fastest time to solve a rubik's cube one-handed. *Guinness World Records*, 2023. URL `https://www.guinnessworldrecords.com/world-records/87711-fastest-time-to-solve-a-rubiks-cube-one-handed`. [Online; accessed 17 June 2023].

[53] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[54] Oliver Scheel, Luca Bergamini, Maciej Wolczyk, Błażej Osiński, and Peter Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *5th Conference on Robot Learning*, pages 718–728, 2021.

[55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[56] Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart, and Sergey Levine. Ving: Learning open-world navigation with visual goals. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13215–13222. IEEE, 2021.

[57] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *6th Annual Conference on Robot Learning*, 2022.

[58] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[59] The Waymo Team. Taking our next step in the city by the bay. *Waymo Blog*, March 2022. URL `https://waymo.com/blog/2022/03/taking-our-next-step-in-city-by-bay.html`.

[60] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[61] Arun Balajee Vasudevan, Dengxin Dai, and Luc Van Gool. Talk2nav: Long-range vision-and-language navigation with dual attention and spatial memory. *International Journal of Computer Vision*, 129:246–266, 2021.

[62] Arun Venkatraman, Martial Hebert, and J Bagnell. Improving multi-step prediction of learned time series models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[63] 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.

[64] Matt Vitelli, Yan Chang, Yawei Ye, Ana Ferreira, Maciej Wołczyk, Błażej Osiński, Moritz Niendorf, Hugo Grimmett, Qiangui Huang, Ashesh Jain, et al. Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 897–904. IEEE, 2022.

Appendix A

Complete Publications

MODEL BASED REINFORCEMENT LEARNING FOR ATARI

Łukasz Kaiser[1,*], **Mohammad Babaeizadeh**[1,*], **Piotr Miłos**[2,3,*], **Błażej Osiński**[2,4,*]
Roy H. Campbell[5], **Konrad Czechowski**[4], **Dumitru Erhan**[1], **Chelsea Finn**[1,6],
Piotr Kozakowski[4], **Sergey Levine**[1], **Afroz Mohiuddin**[1], **Ryan Sepassi**[1],
George Tucker[1], **Henryk Michalewski**[4]
[1]Google Brain, [2]deepsense.ai, [3]Institute of Mathematics of the Polish Academy of Sciences,
[4]Faculty of Mathematics, Informatics and Mechanics, University of Warsaw,
[5]University of Illinois at Urbana–Champaign, [6]Stanford University

ABSTRACT

Model-free reinforcement learning (RL) can be used to learn effective policies for complex tasks, such as Atari games, even from image observations. However, this typically requires very large amounts of interaction – substantially more, in fact, than a human would need to learn the same games. How can people learn so quickly? Part of the answer may be that people can learn how the game works and predict which actions will lead to desirable outcomes. In this paper, we explore how video prediction models can similarly enable agents to solve Atari games with fewer interactions than model-free methods. We describe Simulated Policy Learning (SimPLe), a complete model-based deep RL algorithm based on video prediction models and present a comparison of several model architectures, including a novel architecture that yields the best results in our setting. Our experiments evaluate SimPLe on a range of Atari games in low data regime of 100k interactions between the agent and the environment, which corresponds to two hours of real-time play. In most games SimPLe outperforms state-of-the-art model-free algorithms, in some games by over an order of magnitude.

1 INTRODUCTION

Human players can learn to play Atari games in minutes (Tsividis et al., 2017). However, some of the best model-free reinforcement learning algorithms require tens or hundreds of millions of time steps – the equivalent of several weeks of training in real time. How is it that humans can learn these games so much faster? Perhaps part of the puzzle is that humans possess an intuitive understanding of the physical processes that are represented in the game: we know that planes can fly, balls can roll, and bullets can destroy aliens. We can therefore predict the outcomes of our actions. In this paper, we explore how learned video models can enable learning in the Atari Learning Environment (ALE) benchmark Bellemare et al. (2015); Machado et al. (2018) with a budget restricted to 100K time steps – roughly to two hours of a play time.

Although prior works have proposed training predictive models for next-frame, future-frame, as well as combined future-frame and reward predictions in Atari games (Oh et al. (2015); Chiappa et al. (2017); Leibfried et al. (2016)), no prior work has successfully demonstrated model-based control via predictive models that achieve competitive results with model-free RL. Indeed, in a recent survey (Section 7.2 in Machado et al. (2018)) this was formulated as the following challenge: "*So far, there has been no clear demonstration of successful planning with a learned model in the ALE*".

Using models of environments, or informally giving the agent ability to predict its future, has a fundamental appeal for reinforcement learning. The spectrum of possible applications is vast, including learning policies from the model (Watter et al., 2015; Finn et al., 2016; Finn & Levine, 2017; Ebert et al., 2017; Hafner et al., 2019; Piergiovanni et al., 2018; Rybkin et al., 2018; Sutton & Barto,

[*]Equal contribution, authors listed in random order. BO performed the work partially during an internship at Google Brain. Correspondence to: b.osinski@mimuw.edu.pl

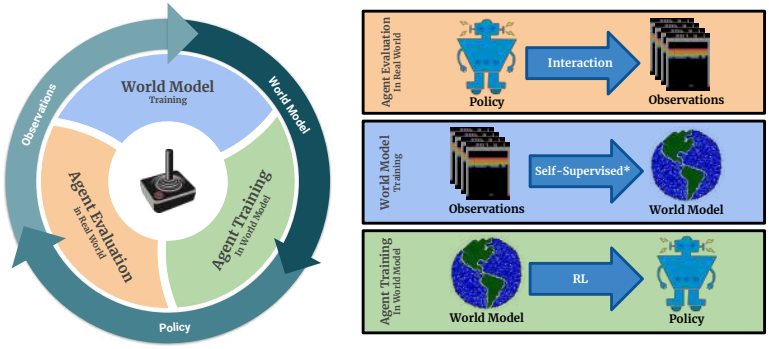


Figure 1: Main loop of SimPLe. 1) the agent starts interacting with the real environment following the latest policy (initialized to random). 2) the collected observations will be used to train (update) the current world model. 3) the agent updates the policy by acting inside the world model. The new policy will be evaluated to measure the performance of the agent as well as collecting more data (back to 1). Note that world model training is self-supervised for the observed states and supervised for the reward.

2017, Chapter 8), capturing important details of the scene (Ha & Schmidhuber, 2018), encouraging exploration (Oh et al., 2015), creating intrinsic motivation (Schmidhuber, 2010) or counterfactual reasoning (Buesing et al., 2019). One of the exciting benefits of model-based learning is the promise to substantially improve sample efficiency of deep reinforcement learning (see Chapter 8 in Sutton & Barto (2017)).

Our work advances the state-of-the-art in model-based reinforcement learning by introducing a system that, to our knowledge, is the first to successfully handle a variety of challenging games in the ALE benchmark. To that end, we experiment with several stochastic video prediction techniques, including a novel model based on discrete latent variables. We present an approach, called Simulated Policy Learning (SimPLe), that utilizes these video prediction techniques and trains a policy to play the game within the learned model. With several iterations of dataset aggregation, where the policy is deployed to collect more data in the original game, we learn a policy that, for many games, successfully plays the game in the real environment (see videos on the project webpage `https://goo.gl/itykP8`).

In our empirical evaluation, we find that SimPLe is significantly more sample-efficient than a highly tuned version of the state-of-the-art Rainbow algorithm (Hessel et al., 2018) on almost all games. In particular, in low data regime of 100k samples, on more than half of the games, our method achieves a score which Rainbow requires at least twice as many samples. In the best case of `Freeway`, our method is more than 10x more sample-efficient, see Figure 3. Since the publication of the first preprint of this work, it has been shown in van Hasselt et al. (2019); Kielak (2020) that Rainbow can be tuned to have better results in low data regime. The results are on a par with SimPLe – both of the model-free methods are better in 13 games, while SimPLe is better in the other 13 out of the total 26 games tested (note that in Section 4.2 van Hasselt et al. (2019) compares with the results of our first preprint, later improved).

2 Related Work

Atari games gained prominence as a benchmark for reinforcement learning with the introduction of the Arcade Learning Environment (ALE) Bellemare et al. (2015). The combination of reinforcement learning and deep models then enabled RL algorithms to learn to play Atari games directly from images of the game screen, using variants of the DQN algorithm (Mnih et al., 2013; 2015; Hessel et al., 2018) and actor-critic algorithms (Mnih et al., 2016; Schulman et al., 2017; Babaeizadeh et al., 2017b; Wu et al., 2017; Espeholt et al., 2018). The most successful methods in this domain remain model-free algorithms (Hessel et al., 2018; Espeholt et al., 2018). Although the sample complexity of these methods has substantially improved recently, it remains far higher than the amount of experience required for human players to learn each game (Tsividis et al., 2017). In this work, we aim to learn Atari games with a budget of just 100K agent steps (400K frames), corresponding to about two hours of play time. Prior methods are generally not evaluated in this regime, and we therefore optimized Rainbow (Hessel et al., 2018) for optimal performance on 1M steps, see Appendix E for details.

Oh et al. (2015) and Chiappa et al. (2017) show that learning predictive models of Atari 2600 environments is possible using appropriately chosen deep learning architectures. Impressively, in some cases the predictions maintain low L_2 error over timespans of hundreds of steps. As learned simulators of Atari environments are core ingredients of our approach, in many aspects our work is motivated by Oh et al. (2015) and Chiappa et al. (2017), however we focus on using video prediction in the context of learning how to play the game well and positively verify that learned simulators can be used to train a policy useful in original environments. An important step in this direction was made by Leibfried et al. (2016), which extends the work of Oh et al. (2015) by including reward prediction, but does not use the model to learn policies that play the games. Most of these approaches, including ours, encode knowledge of the game in implicit way. Unlike this, there are works in which modeling is more explicit, for example Ersen & Sariel (2014) uses testbed of the Incredible Machines to learn objects behaviors and their interactions. Similarly Guzdial et al. (2017) learns an engine predicting interactions of predefined set of sprites in the domain of Super Mario Bros.

Perhaps surprisingly, there is virtually no work on model-based RL in video games from images. Notable exceptions are the works of Oh et al. (2017), Sodhani et al. (2019), Ha & Schmidhuber (2018), Holland et al. (2018), Leibfried et al. (2018) and Azizzadenesheli et al. (2018). Oh et al. (2017) use a model of rewards to augment model-free learning with good results on a number of Atari games. However, this method does not actually aim to model or predict future frames, and achieves clear but relatively modest gains in efficiency. Sodhani et al. (2019) proposes learning a model consistent with RNN policy which helps to train policies that are more powerful than their model-free baseline. Ha & Schmidhuber (2018) present a way to compose a variational autoencoder with a recurrent neural network into an architecture that is successfully evaluated in the VizDoom environment and on a 2D racing game. The training procedure is similar to Algorithm 1, but only one iteration of the loop is needed as the environments are simple enough to be fully explored with random exploration. Similarly, Alaniz (2018) utilizes a transition model with Monte Carlo tree search to solve a block-placing task in Minecraft. Holland et al. (2018) use a variant of Dyna (Sutton, 1991) to learn a model of the environment and generate experience for policy training in the context of Atari games. Using six Atari games as a benchmark Holland et al. (2018) measure the impact of planning shapes on performance of the Dyna-DQN algorithm and include ablations comparing scores obtained with perfect and imperfect models. Our method achieves around 330% of the Dyna-DQN score on Asterix, 120% on Q-Bert, 150% on Seaquest and 80% on Ms. Pac-Man. Azizzadenesheli et al. (2018) propose an algorithm called Generative Adversarial Tree Search (GATS) and for five Atari games train a GAN-based world model along with a Q-function. Azizzadenesheli et al. (2018) primarily discuss various failure modes of the GATS algorithm. Our method achieves around 64 times the score of GATS on Pong and 10 times on Breakout. [1]

Outside of games, model-based reinforcement learning has been investigated at length for applications such as robotics (Deisenroth et al., 2013). Though most of such works do not use image observations, several recent works have incorporated images into real-world (Finn et al., 2016; Finn & Levine, 2017; Babaeizadeh et al., 2017a; Ebert et al., 2017; Piergiovanni et al., 2018; Paxton et al., 2019; Rybkin et al., 2018; Ebert et al., 2018) and simulated (Watter et al., 2015; Hafner et al., 2019) robotic control. Our video models of Atari environments described in Section 4 are motivated by models developed in the context of robotics. Another source of inspiration are discrete autoencoders proposed by van den Oord et al. (2017) and Kaiser & Bengio (2018).

The structure of the model-based RL algorithm that we employ consists of alternating between learning a model, and then using this model to optimize a policy with model-free reinforcement learning. Variants of this basic algorithm have been proposed in a number of prior works, starting from Dyna Q Sutton (1991) to more recent methods that incorporate deep networks Heess et al. (2015); Feinberg et al. (2018); Kalweit & Boedecker (2017); Kurutach et al. (2018).

[1] Comparison with Dyna-DQN and GATS is based on random-normalized scores achieved at 100K interactions. Those are approximate, as the authors Dyna-DQN and GATS have not provided tabular results. Authors of Dyna-DQN also report scores on two games which we do not consider: Beam Rider and Space Invaders. For both games the reported scores are close to random scores, as are GATS scores on Asterix.

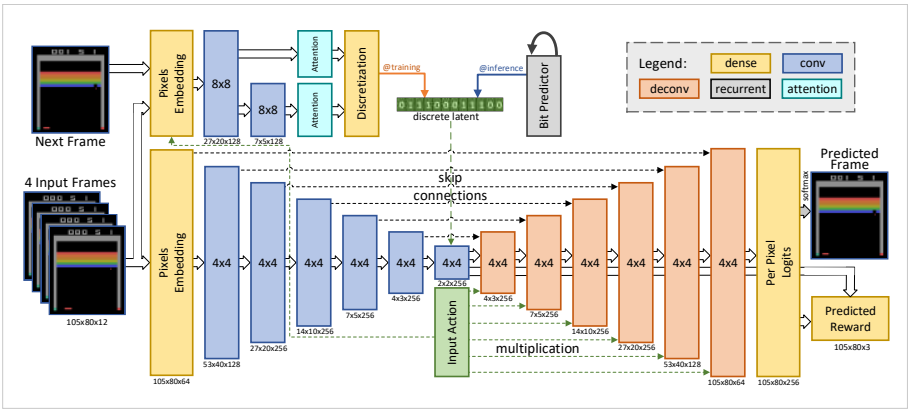


Figure 2: Architecture of the proposed stochastic model with discrete latent. The input to the model is four stacked frames (as well as the action selected by the agent) while the output is the next predicted frame and expected reward. Input pixels and action are embedded using fully connected layers, and there is per-pixel softmax (256 colors) in the output. This model has two main components. First, the bottom part of the network which consists of a skip-connected convolutional encoder and decoder. To condition the output on the actions of the agent, the output of each layer in the decoder is multiplied with the (learned) embedded action. Second part of the model is a convolutional inference network which approximates the posterior given the next frame, similarly to Babaeizadeh et al. (2017a). At training time, the sampled latent values from the approximated posterior will be discretized into bits. To keep the model differentiable, the backpropagation bypasses the discretization following Kaiser & Bengio (2018). A third LSTM based network is trained to approximate each bit given the previous ones. At inference time, the latent bits are predicted auto-regressively using this network. The deterministic model has the same architecture as this figure but without the inference network.

3 Simulated Policy Learning (SimPLe)

Reinforcement learning is formalized in Markov decision processes (MDP). An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is a set of actions available to an agent, P is the unknown transition kernel, r is the reward function and $\gamma \in (0, 1)$ is the discount factor. In this work we refer to MDPs as environments and assume that environments do not provide direct access to the state (i.e., the RAM of Atari 2600 emulator). Instead we use visual observations, typically 210×160 RGB images. A single image does not determine the state. In order to reduce environment's partial observability, we stack four consecutive frames and use it as the observation. A reinforcement learning agent interacts with the MDP by issuing actions according to a policy. Formally, policy π is a mapping from states to probability distributions over $\mathcal{A}$. The quality of a policy is measured by the value function $\mathbb{E}_\pi\left(\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = s\right)$, which for a starting state s estimates the total discounted reward gathered by the agent.

In Atari 2600 games our goal is to find a policy which maximizes the value function from the beginning of the game. Crucially, apart from an Atari 2600 emulator environment env we will use *a neural network simulated environment* env' which we call a *world model* and describe in detail in Section 4. The environment env' shares the action space and reward space with env and produces visual observations in the same format, as it will be trained to mimic env. Our principal aim is to train a policy π using a simulated environment env' so that π achieves good performance in the original environment env. In this training process we aim to use as few interactions with env as possible. The initial data to train env' comes from random rollouts of env. As this is unlikely to capture all aspects of env, we use the iterative method presented in Algorithm 1.

Algorithm 1: Pseudocode for SimPLe

```
Initialize policy π
Initialize model parameters θ of env'
Initialize empty set D
while not done do
    ▷ collect observations from real env.
    D ← D ∪ COLLECT(env, π)
    ▷ update model using collected data.
    θ ← TRAIN_SUPERVISED(env', D)
    ▷ update policy using world model.
    π ← TRAIN_RL(π, env')
end while
```

4 WORLD MODELS

In search for an effective world model we experimented with various architectures, both new and modified versions of existing ones. This search resulted in a novel stochastic video prediction model (visualized in Figure 2) which achieved superior results compared to other previously proposed models. In this section, we describe the details of this architecture and the rationale behind our design decisions. In Section 6 we compare the performance of these models.

Deterministic Model. Our basic architecture, presented as part of Figure 2, resembles the convolutional feedforward network from Oh et al. (2015). The input X consists of four consecutive game frames and an action a. Stacked convolution layers process the visual input. The actions are one-hot-encoded and embedded in a vector which is multiplied channel-wise with the output of the convolutional layers. The network outputs the next frame of the game and the value of the reward.

In our experiments, we varied details of the architecture above. In most cases, we use a stack of four convolutional layers with 64 filters followed by three dense layers (the first two have 1024 neurons). The dense layers are concatenated with 64 dimensional vector with a learnable action embedding. Next, three deconvolutional layers of 64 filters follow. An additional deconvolutional layer outputs an image of the original 105×80 size. The number of filters is either 3 or 3×256. In the first case, the output is a real-valued approximation of pixel's RGB value. In the second case, filters are followed by softmax producing a probability distribution on the color space. The reward is predicted by a softmax attached to the last fully connected layer. We used dropout equal to 0.2 and layer normalization.

Loss functions. The visual output of our networks is either one float per pixel/channel or the categorical 256-dimensional softmax. In both cases, we used the *clipped loss* $\max(Loss, C)$ for a constant C. We found that clipping was crucial for improving the models (measured with the correct reward predictions per sequence metric and successful training using Algorithm 1). We conjecture that clipping substantially decreases the magnitude of gradients stemming from fine-tuning of big areas of background consequently letting the optimization process concentrate on small but important areas (e.g. the ball in Pong). In our experiments, we set $C = 10$ for L_2 loss on pixel values and to $C = 0.03$ for softmax loss. Note that this means that when the level of confidence about the correct pixel value exceeds 97% (as $-\ln(0.97) \approx 0.03$) we get no gradients from that pixel any longer.

Scheduled sampling. The model env' consumes its own predictions from previous steps and due to compounding errors, the model may drift out of the area of its applicability. Following Bengio et al. (2015); Venkatraman et al. (2016), we mitigate this problem by randomly replacing in training some frames of the input X by the prediction from the previous step while linearly increasing the mixing probability to 100% around the middle of the first iteration of the training loop.

Stochastic Models. A stochastic model can be used to deal with limited horizon of past observed frames as well as sprites occlusion and flickering which results to higher quality predictions. Inspired by Babaeizadeh et al. (2017a), we tried a variational autoencoder (Kingma & Welling, 2014) to model the stochasticity of the environment. In this model, an additional network receives the input frames as well as the future target frame as input and approximates the distribution of the posterior. At each timestep, a latent value z_t is sampled from this distribution and passed as input to the original predictive model. At test time, the latent values are sampled from an assumed prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$. To match the assumed prior and the approximate, we use the Kullback–Leibler divergence term as an additional loss term (Babaeizadeh et al., 2017a).

We noticed two major issues with the above model. First, the weight of the KL divergence loss term is game dependent, which is not practical if one wants to deal with a broad portfolio of Atari games. Second, this weight is usually a very small number in the range of $[10^{-3}, 10^{-5}]$ which means that the approximated posterior can diverge significantly from the assumed prior. This can result in previously unseen latent values at inference time that lead to poor predictions. We address these issues by utilizing a discrete latent variable similar to Kaiser & Bengio (2018).

As visualized in Figure 2, the proposed stochastic model with discrete latent variables discretizes the latent values into bits (zeros and ones) while training an auxiliary LSTM-based Hochreiter & Schmidhuber (1997) recurrent network to predict these bits autoregressively. At inference time, the latent bits will be generated by this auxiliary network in contrast to sampling from a prior. To make the predictive model more robust to unseen latent bits, we add uniform noise to approximated latent

values before discretization and apply dropout (Srivastava et al., 2014) on bits after discretization. More details about the architecture is in Appendix C.

5 POLICY TRAINING

We will now describe the details of SimPLe, outlined in Algorithm 1. In step 6 we use the proximal policy optimization (PPO) algorithm (Schulman et al., 2017) with $\gamma = 0.95$. The algorithm generates rollouts in the simulated environment env' and uses them to improve policy π. The fundamental difficulty lays in imperfections of the model compounding over time. To mitigate this problem we use short rollouts of env'. Typically every $N = 50$ steps we uniformly sample the starting state from the ground-truth buffer D and restart env' (for experiments with the value of γ and N see Section 6.4). Using short rollouts may have a degrading effect as the PPO algorithm does not have a way to infer effects longer than the rollout length. To ease this problem, in the last step of a rollout we add to the reward the evaluation of the value function. Training with multiple iterations re-starting from trajectories gathered in the real environment is new to our knowledge. It was inspired by the classical Dyna-Q algorithm and, notably, in the Atari domain no comparable results have been achieved.

The main loop in Algorithm 1 is iterated 15 times (cf. Section 6.4). The world model is trained for 45K steps in the first iteration and for 15K steps in each of the following ones. Shorter training in later iterations does not degrade the performance because the world model after first iteration captures already part of the game dynamics and only needs to be extended to novel situations.

In each of the iterations, the agent is trained inside the latest world model using PPO. In every PPO epoch we used 16 parallel agents collecting 25, 50 or 100 steps from the simulated environment env' (see Section 6.4 for ablations). The number of PPO epochs is $z \cdot 1000$, where z equals to 1 in all passes except last one (where $z = 3$) and two passes number 8 and 12 (where $z = 2$). This gives $800\text{K}{\cdot}z$ interactions with the simulated environment in each of the loop passes. In the process of training the agent performs 15.2M interactions with the simulated environment env'.

6 EXPERIMENTS

We evaluate SimPLe on a suite of Atari games from Atari Learning Environment (ALE) benchmark. In our experiments, the training loop is repeated for 15 iterations, with 6400 interactions with the environment collected in each iteration. We apply a standard pre-processing for Atari games: a frame skip equal to 4, that is every action is repeated 4 times. The frames are down-scaled by a factor of 2.

Because some data is collected before the first iteration of the loop, altogether $6400 \cdot 16 = 102,400$ interactions with the Atari environment are used during training. This is equivalent to $409,600$ frames from the Atari game (114 minutes at 60 FPS). At every iteration, the latest policy trained under the learned model is used to collect data in the real environment `env`. The data is also directly used to train the policy with PPO. Due to vast difference between number of training data from simulated environment and real environment (15M vs 100K) the impact of the latter on policy is negligible.

We evaluate our method on 26 games selected on the basis of being solvable with existing state-of-the-art model-free deep RL algorithms[2], which in our comparisons are Rainbow Hessel et al. (2018) and PPO Schulman et al. (2017). For Rainbow, we used the implementation from the Dopamine package and spent considerable time tuning it for sample efficiency (see Appendix E).

For visualization of all experiments see `https://goo.gl/itykP8` and for a summary see Figure 3. It can be seen that our method is more sample-efficient than a highly tuned Rainbow baseline on almost all games, requires less than half of the samples on more than half of the games and, on `Freeway`, is more than 10x more sample-efficient. Our method outperforms PPO by an even larger margin. We also compare our method with fixed score baselines (for different baselines) rather than counting how many steps are required to match our score, see Figure 4 for the results. For the

[2]Specifically, for the final evaluation we selected games which achieved non-random results using our method or the Rainbow algorithm using 100K interactions.

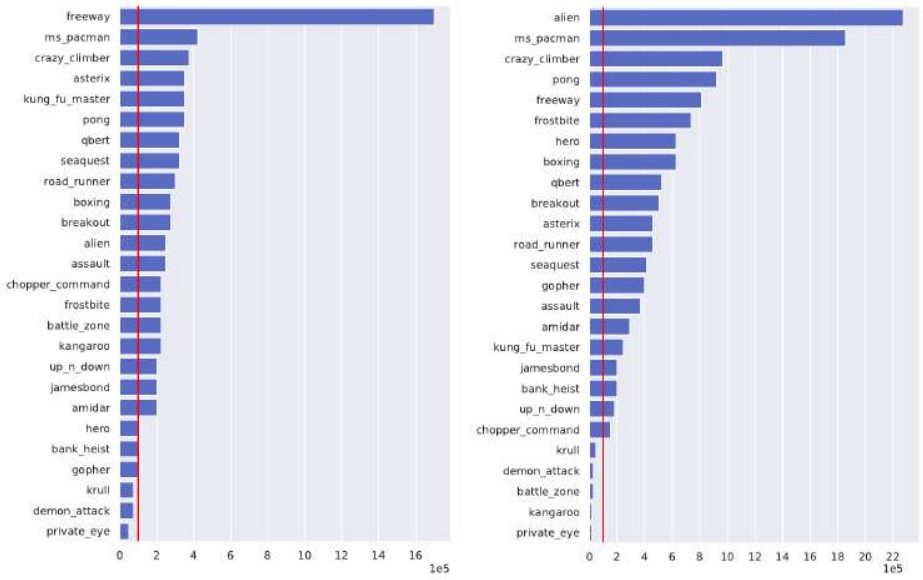


Figure 3: Comparison with Rainbow and PPO. Each bar illustrates the number of interactions with environment required by Rainbow (left) or PPO (right) to achieve the same score as our method (SimPLe). The red line indicates the 100*K interactions threshold which is used by the our method.*

qualitative analysis of performance on different games see Appendix B. The source code is available as part of the Tensor2Tensor library and it includes instructions on how to run the experiments[3].

6.1 Sample Efficiency

The primary evaluation in our experiments studies the sample efficiency of SimPLe, in comparison with state-of-the-art model-free deep RL methods in the literature. To that end, we compare with Rainbow (Hessel et al., 2018; Castro et al., 2018), which represents the state-of-the-art Q-learning method for Atari games, and PPO (Schulman et al., 2017), a model-free policy gradient algorithm (see Appendix E for details of tuning of Rainbow and PPO). The results of the comparison are presented in Figure 3. For each game, we plot the number of time steps needed for either Rainbow or PPO to reach the same score that our method reaches after 100K interaction steps. The red line indicates 100K steps: any bar larger than this indicates a game where the model-free method required more steps. SimPLe outperforms the model-free algorithms in terms of learning speed on nearly all of the games, and in the case of a few games, does so by over an order of magnitude. For some games, it reaches the same performance that our PPO implementation reaches at 10M steps. This indicates that model-based reinforcement learning provides an effective approach to learning Atari games, at a fraction of the sample complexity.

The results in these figures are generated by averaging 5 runs for each game. The model-based agent is better than a random policy for all the games except `Bank Heist`. Interestingly, we observed that the best of the 5 runs was often significantly better. For 6 of the games, it exceeds the average human score (as reported in Table 3 of Pohlen et al. (2018)). This suggests that further stabilizing SimPLe should improve its performance, indicating an important direction for future work. In some cases during training we observed high variance of the results during each step of the loop. There are a number of possible reasons, such as mutual interactions of the policy training and the supervised training or domain mismatch between the model and the real environment. We present detailed numerical results, including best scores and standard deviations, in Appendix D.

[3] `https://github.com/tensorflow/tensor2tensor/tree/master/tensor2tensor/rl`

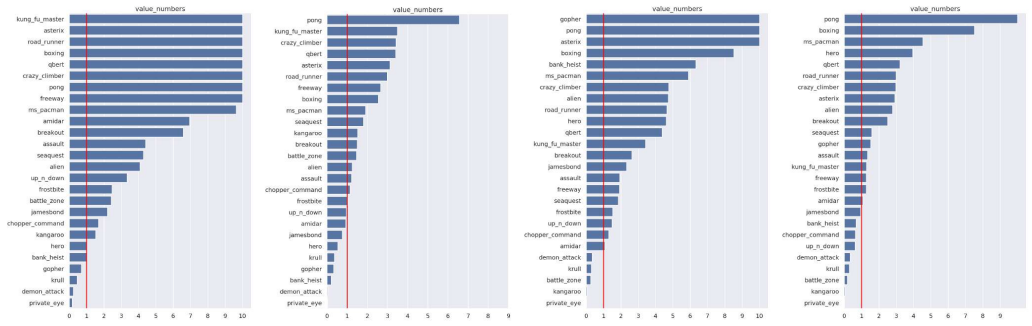
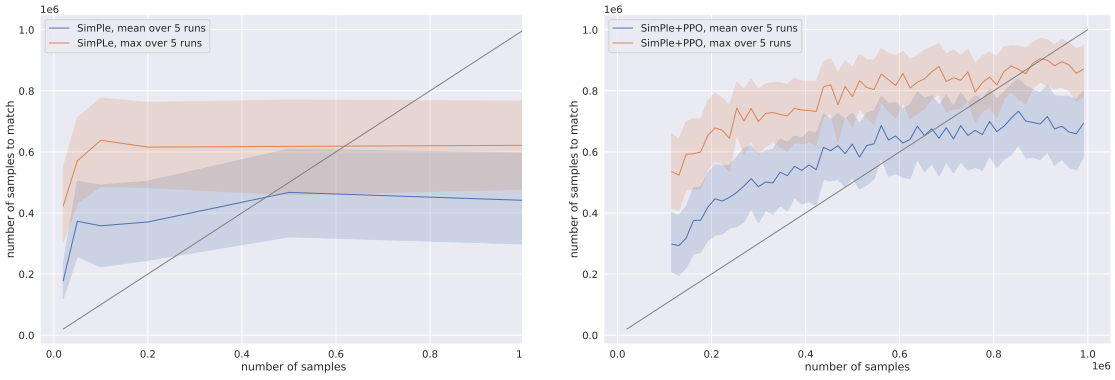


Figure 4: Fractions of Rainbow and PPO scores at different numbers of interactions calculated with the formula $(SimPLe\_score@100K - random\_score)/(baseline\_score - random\_score)$*; if denominator is smaller than 0, both nominator and denominator are increased by 1. From left to right, the baselines are: Rainbow at 100K, Rainbow at 200K, PPO at 100K, PPO at 200K. SimPLe outperforms Rainbow and PPO even when those are given twice as many interactions.*



(a) Model based (SimPLe)

(b) Model based + model free (SimPLe + PPO)

Figure 5: Behaviour with respect to the number of used samples. We report number of frames required by PPO to reach the score of our models. Results are averaged over all games.

6.2 Number of frames

We focused our work on learning games with 100K interaction steps with the environment. In this section we present additional results for settings with 20K, 50K, 200K, 500K and 1M interactions; see Figure 5 (a). Our results are poor with 20K interactions. For 50K they are already almost as good as with 100K interactions. From there the results improve until 500K samples – it is also the point at which they are on par with model-free PPO. Detailed per game results can be found in Appendix F.

This demonstrates that SimPLe excels in a low data regime, but its advantage disappears with a bigger amount of data. Such a behavior, with fast growth at the beginning of training, but lower asymptotic performance is commonly observed when comparing model-based and model-free methods (Wang et al. (2019)). As observed in Section 6.4 assigning bigger computational budget helps in 100K setting. We suspect that gains would be even bigger for the settings with more samples.

Finally, we verified if a model obtained with SimPLe using 100K is a useful initialization for model-free PPO training. Based on the results depicted in Figure 5 (b) we can positively answer this conjecture. Lower asymptotic performance is probably due to worse exploration. A policy pre-trained with SimPLe was meant to obtain the best performance on 100K, at which point its entropy is very low thus hindering further PPO training.

6.3 ENVIRONMENT STOCHASTICITY

A crucial decision in the design of world models is the inclusion of stochasticity. Although Atari is known to be a deterministic environment, it is stochastic given only a limited horizon of past observed frames (in our case 4 frames). The level of stochasticity is game dependent; however, it can be observed in many Atari games. An example of such behavior can be observed in the game `Kung Fu Master` – after eliminating the current set of opponents, the game screen always looks the same (it contains only player's character and the background). The game dispatches diverse sets of new opponents, which cannot be inferred from the visual observation alone (without access to the game's internal state) and thus cannot be predicted by a deterministic model. Similar issues have been reported in Babaeizadeh et al. (2017a), where the output of their baseline deterministic model was a blurred superposition of possible random object movements. As can be seen in Figure 11 in the Appendix, the stochastic model learns a reasonable behavior – samples potential opponents and renders them sharply.

Given the stochasticity of the proposed model, SimPLe can be used with truly stochastic environments. To demonstrate this, we ran an experiment where the full pipeline (both the world model and the policy) was trained in the presence of sticky actions, as recommended in (Machado et al., 2018, Section 5). Our world model learned to account for the stickiness of actions and in most cases the end results were very similar to the ones for the deterministic case even without any tuning, see Figure 6.

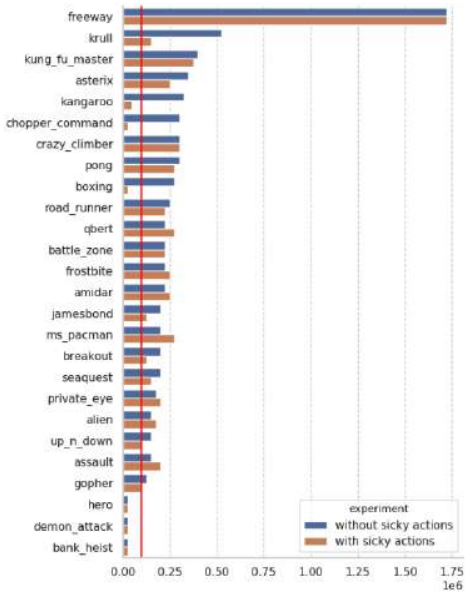


Figure 6: Impact of the environment stochasticity. The graphs are in the same format as Figure 3: each bar illustrates the number of interactions with environment required by Rainbow to achieve the same score as SimPLe (with stochastic discrete world model) using 100k steps in an environment with and without sticky actions.

6.4 ABLATIONS

To evaluate the design of our method, we independently varied a number of the design decisions. Here we present an overview; see Appendix A for detailed results.

Model architecture and hyperparameters. We evaluated a few choices for the world model and our proposed stochastic discrete model performs best by a significant margin. The second most important parameter was the length of world model's training. We verified that a longer training would be beneficial, however we had to restrict it in all other ablation studies due to a high cost of training on all games. As for the length of rollouts from simulated env', we use $N = 50$ by default. We experimentally shown that $N = 25$ performs roughly on par, while $N = 100$ is slightly worse, likely due to compounding model errors. The *discount factor* was set to $\gamma = 0.99$ unless specified otherwise. We see that $\gamma = 0.95$ is slightly better than other values, and we hypothesize that it is due to better tolerance to model imperfections. But overall, all three values of γ perform comparably.

Model-based iterations. The iterative process of training the model, training the policy, and collecting data is crucial for non-trivial tasks where random data collection is insufficient. In a game-by-game analysis, we quantified the number of games where the best results were obtained in later iterations of training. In some games, good policies could be learned very early. While this might have been due to the high variability of training, it does suggest the possibility of much faster training (i.e. in fewer step than 100k) with more directed exploration policies. In Figure 9 in the Appendix we present the cumulative distribution plot for the (first) point during learning when the maximum score for the run was achieved in the main training loop of Algorithm 1.

Random starts. Using short rollouts is crucial to mitigate the compounding errors in the model. To ensure exploration, SimPLe starts rollouts from randomly selected states taken from the real data buffer D. Figure 9 compares the baseline with an experiment without random starts and rollouts of length 1000 on `Seaquest` which shows much worse results without random starts.

7 CONCLUSIONS AND FUTURE WORK

We presented SimPLe, a model-based reinforcement learning approach that operates directly on raw pixel observations and learns effective policies to play games in the Atari Learning Environment. Our experiments demonstrate that SimPLe learns to play many of the games with just 100K interactions with the environment, corresponding to 2 hours of play time. In many cases, the number of samples required for prior methods to learn to reach the same reward value is several times larger.

Our predictive model has stochastic latent variables so it can be applied in highly stochastic environments. Studying such environments is an exciting direction for future work, as is the study of other ways in which the predictive neural network model could be used. Our approach uses the model as a learned simulator and directly applies model-free policy learning to acquire the policy. However, we could use the model for planning. Also, since our model is differentiable, the additional information contained in its gradients could be incorporated into the reinforcement learning process. Finally, the representation learned by the predictive model is likely be more meaningful by itself than the raw pixel observations from the environment. Incorporating this representation into the policy could further accelerate and improve the reinforcement learning process.

While SimPLe is able to learn more quickly than model-free methods, it does have limitations. First, the final scores are on the whole lower than the best state-of-the-art model-free methods. This can be improved with better dynamics models and, while generally common with model-based RL algorithms, suggests an important direction for future work. Another, less obvious limitation is that the performance of our method generally varied substantially between different runs on the same game. The complex interactions between the model, policy, and data collection were likely responsible for this. In future work, models that capture uncertainty via Bayesian parameter posteriors or ensembles (Kurutach et al., 2018; Chua et al., 2018) may improve robustness. Finally, the computational and time requirement of training inside world model are substantial (see Appendix C), which makes developing lighter models an important research direction.

In this paper our focus was to demonstrate the capability and generality of SimPLe only across a suite of Atari games, however, we believe similar methods can be applied to other environments and tasks which is one of our main directions for future work. As a long-term challenge, we believe that model-based reinforcement learning based on stochastic predictive models represents a promising and highly efficient alternative to model-free RL. Applications of such approaches to both high-fidelity simulated environments and real-world data represent an exciting direction for future work that can enable highly efficient learning of behaviors from raw sensory inputs in domains such as robotics and autonomous driving.

ACKNOWLEDGMENTS

We thank Marc Bellemare and Pablo Castro for their help with Rainbow and Dopamine. The work of Konrad Czechowski, Piotr Kozakowski and Piotr Miłoś was supported by the Polish National Science Center grants UMO-2017/26/E/ST6/00622. The work of Henryk Michalewski was supported by the Polish National Science Center grant UMO-2018/29/B/ST6/02959. This research was supported by the PL-Grid Infrastructure. In particular, Konrad Czechowski, Piotr Kozakowski, Henryk Michalewski, Piotr Miłoś and Błażej Osiński extensively used the Prometheus supercomputer, located in the Academic Computer Center Cyfronet in the AGH University of Science and Technology in Kraków, Poland. Some of the experiments were managed using `https://neptune.ai`. We would like to thank the Neptune team for providing us access to the team version and technical support.

REFERENCES

Stephan Alaniz. Deep reinforcement learning with model learning and monte carlo tree search in minecraft. *arXiv preprint arXiv:1803.08456*, 2018.

Kamyar Azizzadenesheli, Brandon Yang, Weitang Liu, Emma Brunskill, Zachary C. Lipton, and Animashree Anandkumar. Sample-efficient deep RL with generative adversarial tree search. *CoRR*, abs/1806.05780, 2018.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. *ICLR*, 2017a.

Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a GPU. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017b. URL https://openreview.net/forum?id=r1VGvBcxl.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents (extended abstract). In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 4148–4152, 2015.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1171–1179, 2015.

Lars Buesing, Theophane Weber, Yori Zwols, Nicolas Heess, Sébastien Racanière, Arthur Guez, and Jean-Baptiste Lespiau. Woulda, coulda, shoulda: Counterfactually-guided policy search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=BJG0voC9YQ.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A research framework for deep reinforcement learning. *CoRR*, abs/1812.06110, 2018.

Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=B1s6xvqlx.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4759–4770, 2018.

Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2), 2013.

Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pp. 344–356. PMLR, 2017.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Mustafa Ersen and Sanem Sariel. Learning behaviors of and interactions among objects through spatio–temporal reasoning. *IEEE Transactions on Computational Intelligence and AI in Games*, 7 (1):75–87, 2014.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pp. 1406–1415, 2018.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101, 2018.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pp. 2786–2793. IEEE, 2017. doi: 10.1109/ICRA.2017.7989324.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *IEEE International Conference on Robotics and Automation, ICRA*, pp. 512–519, 2016.

Matthew Guzdial, Boyang Li, and Mark O. Riedl. Game engine learning from video. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pp. 3707–3713, 2017. doi: 10.24963/ijcai.2017/518.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 2455–2467, 2018.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565. PMLR, 2019.

Nicolas Heess, Gregory Wayne, David Silver, Timothy P. Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 2944–2952, 2015.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 3215–3222. AAAI Press, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

G. Zacharias Holland, Erik Talvitie, and Michael Bowling. The effect of planning shape on dyna-style planning in high-dimensional state spaces. *CoRR*, abs/1806.01825, 2018.

Lukasz Kaiser and Samy Bengio. Discrete autoencoders for sequence models. *CoRR*, abs/1801.09797, 2018.

Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (eds.), *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 195–206. PMLR, 13–15 Nov 2017.

Kacper Piotr Kielak. Do recent advancements in model-based deep reinforcement learning really improve data efficiency?, 2020. URL `https://openreview.net/forum?id=Bke9u1HFwB`.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=SJJinbWRZ`.

Felix Leibfried, Nate Kushman, and Katja Hofmann. A deep learning approach for joint video frame and reward prediction in Atari games. *CoRR*, abs/1611.07078, 2016.

Felix Leibfried, Rasul Tutunov, Peter Vrancx, and Haitham Bou-Ammar. Model-based regularization for deep reinforcement learning with transcoder networks. *arXiv preprint arXiv:1809.01906*, 2018.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Intell. Res.*, 61:523–562, 2018. doi: 10.1613/jair.5699.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pp. 1928–1937, 2016.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder P. Singh. Action-conditional video prediction using deep networks in atari games. In *NIPS*, pp. 2863–2871, 2015.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6118–6128. Curran Associates, Inc., 2017.

Chris Paxton, Yotam Barnoy, Kapil D. Katyal, Raman Arora, and Gregory D. Hager. Visual robot task planning. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pp. 8832–8838. IEEE, 2019. doi: 10.1109/ICRA.2019.8793736.

A. J. Piergiovanni, Alan Wu, and Michael S. Ryoo. Learning real-world robot policies by dreaming. *CoRR*, abs/1805.07813, 2018.

Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado van Hasselt, John Quan, Mel Vecerík, Matteo Hessel, Rémi Munos, and Olivier Pietquin. Observe and look further: Achieving consistent performance on atari. *CoRR*, abs/1805.11593, 2018.

Oleh Rybkin, Karl Pertsch, Andrew Jaegle, Konstantinos G. Derpanis, and Kostas Daniilidis. Unsupervised learning of sensorimotor affordances by stochastic future prediction. *CoRR*, abs/1806.09655, 2018.

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Trans. Autonomous Mental Development*, 2(3):230–247, 2010.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Shagun Sodhani, Anirudh Goyal, Tristan Deleu, Yoshua Bengio, Sergey Levine, and Jian Tang. Learning powerful policies by using consistent dynamics model. *arXiv preprint arXiv:1906.04355*, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction, 2nd edition (work in progress)*. Adaptive computation and machine learning. MIT Press, 2017.

Pedro Tsividis, Thomas Pouncy, Jaqueline L. Xu, Joshua B. Tenenbaum, and Samuel J. Gershman. Human learning in atari. In *2017 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 27-29, 2017*, 2017.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 6306–6315, 2017.

Hado van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 14322–14333, 2019.

Arun Venkatraman, Roberto Capobianco, Lerrel Pinto, Martial Hebert, Daniele Nardi, and J. Andrew Bagnell. Improved learning of dynamics models for control. In *International Symposium on Experimental Robotics, ISER 2016, Tokyo, Japan, October 3-6, 2016.*, pp. 703–713, 2016.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057, 2019.

Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pp. 2746–2754, 2015.

Yuhuai Wu, Elman Mansimov, Roger B. Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 5279–5288, 2017.

Table 1: ***Summary of SimPLe ablations.*** *For each game, a configuration was assigned a score being the mean over 5 experiments. The best and median scores were calculated per game. The table reports the number of games a given configuration achieved the best score or at least the median score, respectively.*

model	best	at least median
deterministic	0	7
det. recurrent	3	13
SD	8	16
SD $\gamma = 0.9$	1	14
default	10	21
SD 100 steps	0	14
SD 25 steps	4	19

All our code is available as part of the Tensor2Tensor library and it includes instructions on how to run our experiments: `https://github.com/tensorflow/tensor2tensor/tree/master/tensor2tensor/rl`.

A ABLATIONS

To evaluate the design of our method, we independently varied a number of the design decisions: the choice of the model, the γ parameter and the length of PPO rollouts. The results for 7 experimental configurations are summarized in the Table 1.

Models. To assess the model choice, we evaluated the following models: deterministic, deterministic recurrent, and stochastic discrete (see Section 4). Based on Table 1 it can be seen that our proposed stochastic discrete model performs best. Figures 7a and 7b show the role of stochasticity and recurrence.

Steps. See Figure 7d. As described in Section 5 every N steps we reinitialize the simulated environment with ground-truth data. By default we use $N = 50$, in some experiments we set $N = 25$ or $N = 100$. It is clear from the table above and Figure 7d that 100 is a bit worse than either 25 or 50, likely due to compounding model errors, but this effect is much smaller than the effect of model architecture.

Gamma. See Figure 8b. We used the discount factor $\gamma = 0.99$ unless specified otherwise. We see that $\gamma = 0.95$ is slightly better than other values, and we hypothesize that it is due to better tolerance to model imperfections. But overall, all three values of γ seem to perform comparably at the same number of steps.

Model-based iterations. The iterative process of training the model, training the policy, and collecting data is crucial for non-trivial tasks where simple random data collection is insufficient. In the game-by-game analysis, we quantified the number of games where the best results were obtained in later iterations of training. In some games, good policies could be learned very early. While this might have been due simply to the high variability of training, it does suggest the possibility that much faster training – in many fewer than 100k steps – could be obtained in future work with more directed exploration policies. We leave this question to future work.

In Figure 9 we present the cumulative distribution plot for the (first) point during learning when the maximum score for the run was achieved in the main training loop of Algorithm 1.

On Figure 7c we show results for experiments in which the number samples was fixed to be 100K but the number of training loop varied. We conclude that 15 is beneficial for training.

Long model training Our best results were obtained with much 5 times longer training of the world models, see Figure 8a for comparison with shorter training. Due to our resources constraints other ablations were made with the short model training setting.

Random starts. Using short rollouts is crucial to mitigate the compounding errors under the model. To ensure exploration SimPLe starts rollouts from randomly selected states taken from the real data buffer D. In Figure 9 we present a comparison with an experiment without random starts and rollouts of length 1000 on `Seaquest`. These data strongly indicate that ablating random starts substantially deteriorate results.

B QUALITATIVE ANALYSIS

This section provides a qualitative analysis and case studies of individual games. We emphasize that we did not adjust the method nor hyperparameters individually for each game, but we provide specific qualitative analysis to better understand the predictions from the model.[4]

Solved games. The primary goal of our paper was to use model-based methods to achieve good performance within a modest budget of 100k interactions. For two games, `Pong` and `Freeway`, our method, SimPLe, was able to achieve the maximum score.

Exploration. `Freeway` is a particularly interesting game. Though simple, it presents a substantial exploration challenge. The chicken, controlled by the agents, is quite slow to ascend when exploring randomly as it constantly gets bumped down by the cars (see the left video `https://goo.gl/YHbKZ6`). This makes it very unlikely to fully cross the road and obtain a non-zero reward. Nevertheless, SimPLe is able to capture such rare events, internalize them into the predictive model and then successfully learn a successful policy.

However, this good performance did not happen on every run. We conjecture the following scenario in failing cases. If at early stages the entropy of the policy decayed too rapidly the collected experience stayed limited leading to a poor world model, which was not powerful enough to support exploration (e.g. the chicken disappears when moving to high). In one of our experiments, we observed that the final policy was that the chicken moved up only to the second lane and stayed waiting to be hit by the car and so on so forth.

Pixel-perfect games. In some cases (for `Pong`, `Freeway`, `Breakout`) our models were able to predict the future perfectly, down to every pixel. This property holds for rather short time intervals, we observed episodes lasting up to 50 time-steps. Extending it to long sequences would be a very exciting research direction. See videos `https://goo.gl/uyfNnW`.

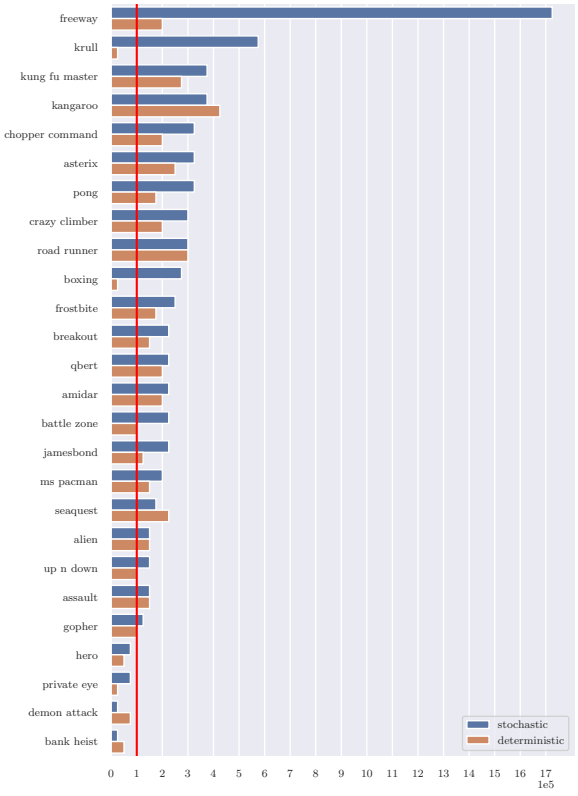
Benign errors. Despite the aforementioned positive examples, accurate models are difficult to acquire for some games, especially at early stages of learning. However, model-based RL should be tolerant to modest model errors. Interestingly, in some cases our models differed from the original games in a way that was harmless or only mildly harmful for policy training.

For example, in `Bowling` and `Pong`, the ball sometimes splits into two. While nonphysical, seemingly these errors did not distort much the objective of the game, see Figure 10 and also `https://goo.gl/JPi7rB`.

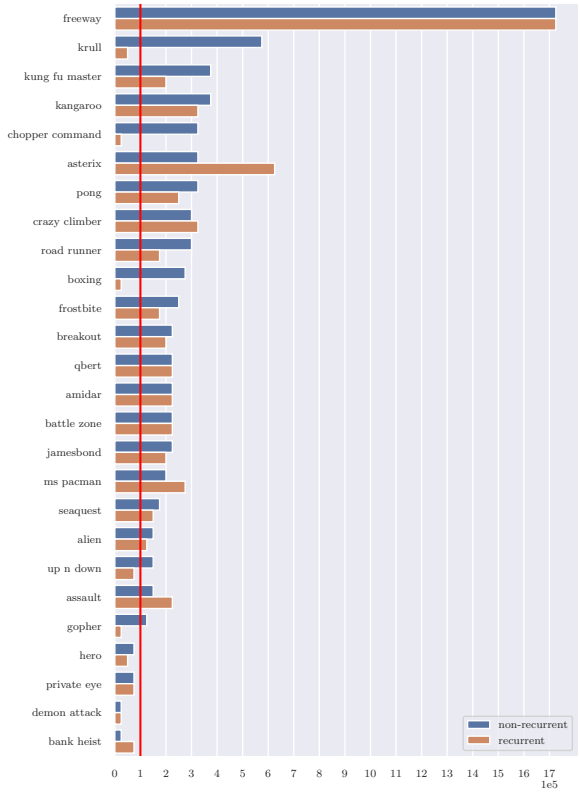
In `Kung Fu Master` our model's predictions deviate from the real game by spawning a different number of opponents, see Figure 11. In `Crazy Climber` we observed the bird appearing earlier in the game. These cases are probably to be attributed to the stochasticity in the model. Though not aligned with the true environment, the predicted behaviors are plausible, and the resulting policy can still play the original game.

Failures on hard games. On some of the games, our models simply failed to produce useful predictions. We believe that listing such errors may be helpful in designing better training protocols and building better models. The most common failure was due to the presence of very small but highly relevant objects. For example, in `Atlantis` and `Battle Zone` bullets are so small that they tend to disappear. Interestingly, `Battle Zone` has pseudo-3D graphics, which may have added to the difficulty. See videos `https://goo.gl/uiccKU`.

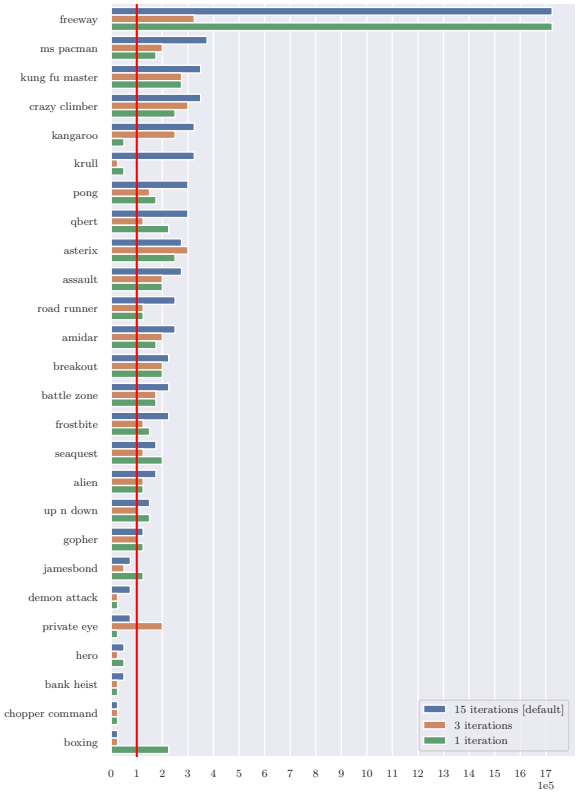
[4] We strongly encourage the reader to watch accompanying videos `https://goo.gl/itykP8`



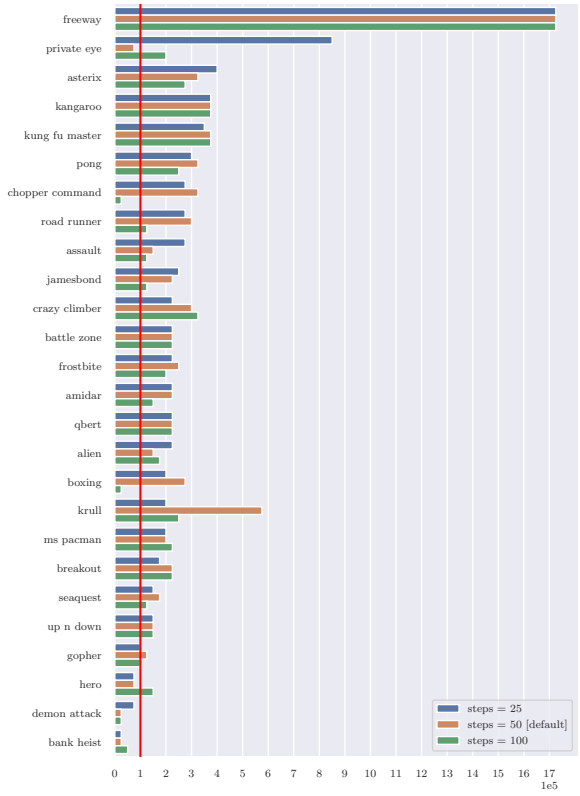
(a) Effect of stochasticity.



(b) Effect of recurrent architecture.

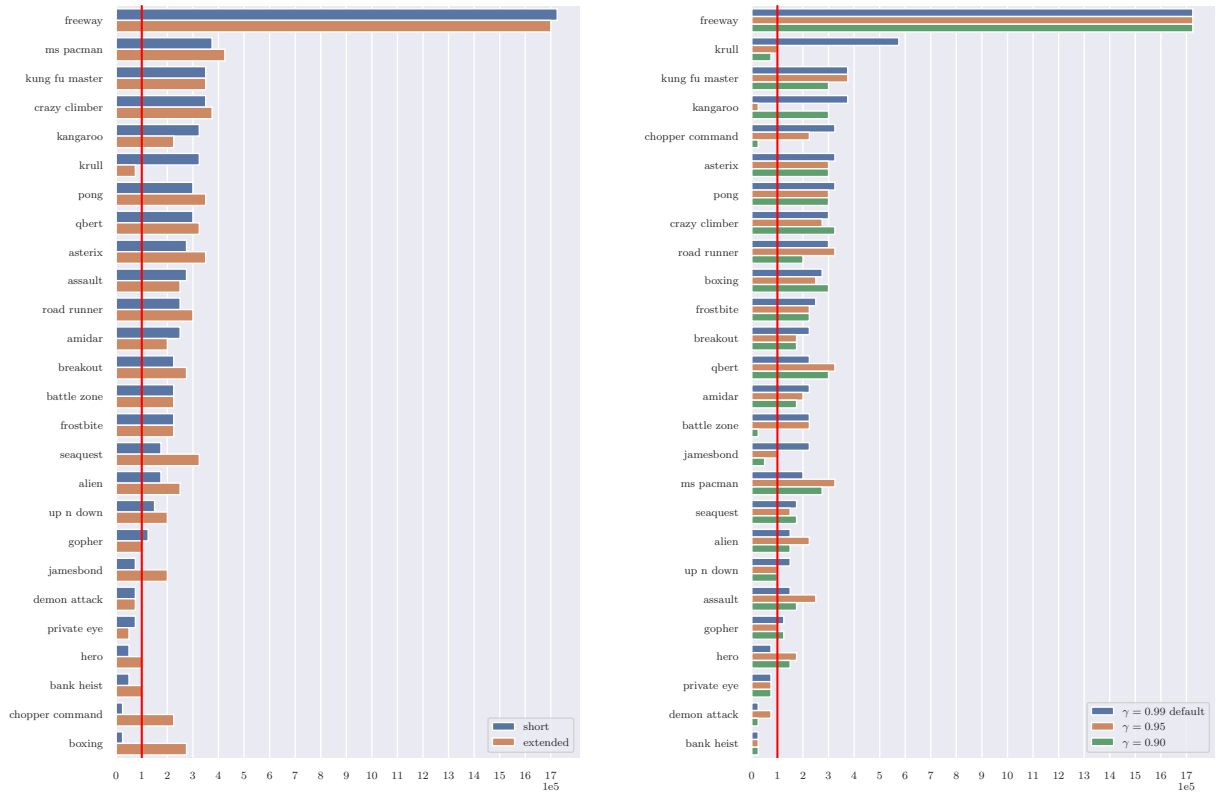


(c) Effect of adjusting of number of epochs.



(d) Effect of adjusting of number of steps.

Figure 7: Ablations part 1. The graphs are in the same format as Figure 3: each bar illustrates the number of interactions with environment required by Rainbow to achieve the same score as a particular variant of SimPLe. The red line indicates the $100K$ *interactions threshold which is used by SimPLe.*



(a) Effect of extended model training.

(b) Effect of adjusting γ in PPO training

Figure 8: Ablations part 2. The graphs are in the same format as Figure 3: each bar illustrates the number of interactions with environment required by Rainbow to achieve the same score as a particular variant of SimPLe. The red line indicates the $100K$ *interactions threshold which is used by SimPLe.*

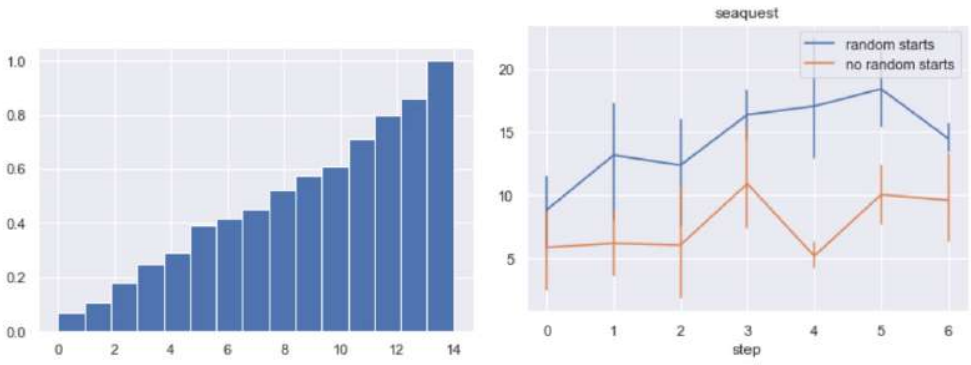


Figure 9: (left) CDF of the number of iterations to acquire maximum score. The vertical axis represents the fraction of all games. (right) Comparison of random starts vs no random starts on `Seaquest` *(for better readability we clip game rewards to* $\{-1, 0, 1\}$*). The vertical axis shows a mean reward and the horizontal axis the number of iterations of Algorithm 1.*

Figure 10: Frames from the `Pong` *environment.*

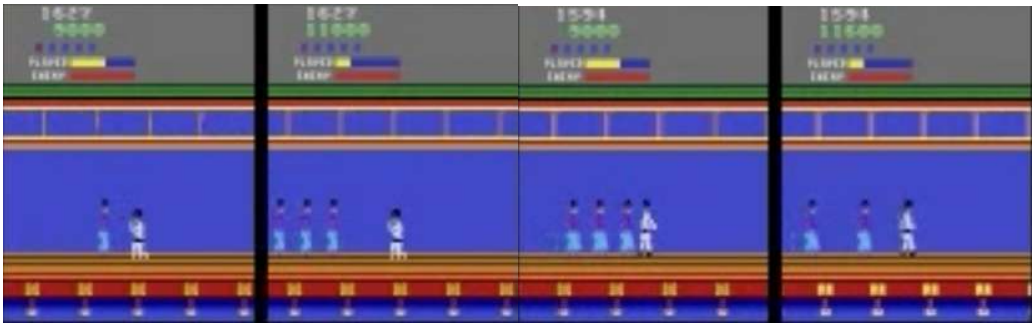


Figure 11: Frames from the `Kung Fu Master` *environment (left) and its model (right).*

Another interesting example comes from `Private Eye` in which the agent traverses different scenes, teleporting from one to the other. We found that our model generally struggled to capture such large global changes.

C ARCHITECTURE DETAILS

The world model is a crucial ingredient of our algorithm. Therefore the neural-network architecture of the model plays a crucial role. The high-level overview of the architecture is given in Section 4 and Figure 2. We stress that the model is general, not Atari specific, and we believe it could handle other visual prediction tasks. The whole model has around 74M parameters and the inference/backpropagation time is approx. 0.5s/0.7s respectively, where inference is on batch size 16 and backpropagation on batch size 2, running on NVIDIA Tesla P100. This gives us around 32ms per frame from our simulator, in comparison one step of the ALE simulator takes approximately 0.4ms.

Below we give more details of the architecture. First, the frame prediction network:

Layer	Number of outputs	Other details
Input frame dense	96	-
Downscale convolution 1	192	kernel 4x4, stride 2x2
Downscale convolution 2	384	kernel 4x4, stride 2x2
Downscale convolution 3	768	kernel 4x4, stride 2x2
Downscale convolution 4	768	kernel 4x4, stride 2x2
Downscale convolution 5	768	kernel 4x4, stride 2x2
Downscale convolution 6	768	kernel 4x4, stride 2x2
Action embedding	768	-
Latent predictor embedding	128	-
Latent predictor LSTM	128	-
Latent predictor output dense	256	-
Reward predictor hidden	128	-
Reward predictor output dense	3	-
Middle convolution 1	768	kernel 3x3, stride 1x1
Middle convolution 2	768	kernel 3x3, stride 1x1
Upscale transposed convolution 1	768	kernel 4x4, stride 2x2
Upscale transposed convolution 2	768	kernel 4x4, stride 2x2
Upscale transposed convolution 3	768	kernel 4x4, stride 2x2
Upscale transposed convolution 4	384	kernel 4x4, stride 2x2
Upscale transposed convolution 5	192	kernel 4x4, stride 2x2
Upscale transposed convolution 6	96	kernel 4x4, stride 2x2
Output frame dense	768	-

The latent inference network, used just during training:

Layer	Number of outputs	Other details
Downscale convolution 1	128	kernel 8x8, stride 4x4
Downscale convolution 2	512	kernel 8x8, stride 4x4

All activation functions are ReLU, except for the layers marked as "output", which have softmax activations, and LSTM internal layers. In the frame prediction network, the downscale layers are connected to the corresponding upscale layers with residual connections. All convolution and transposed convolution layers are preceded by dropout 0.15 and followed by layer normalization. The latent predictor outputs 128 bits sequentially, in chunks of 8.

D NUMERICAL RESULTS

Below we present numerical results of our experiments. We tested SimPLe on 7 configurations (see description in Section A). For each configuration we run 5 experiments. For the evaluation of the i-th experiments we used the policy given by $\text{softmax}(\text{logits}(\pi_i)/T)$, where π_i is the final learnt policy in the experiment and T is the temperature parameter. We found empirically that $T = 0.5$ worked best in most cases. A tentative explanation is that polices with temperatures smaller than 1 are less stochastic and thus more stable. However, going down to $T = 0$ proved to be detrimental in many cases as, possibly, it makes policies more prone to imperfections of models.

In Table 2 we present the mean and standard deviation of the 5 experiments. We observed that the median behaves rather similarly, which is reported it in Table 4. In this table we also show maximal scores over 5 runs. Interestingly, in many cases they turned out to be much higher. This, we hope, indicates that our methods has a further potential of reaching these higher scores.

Human scores are "Avg. Human" from Table 3 in Pohlen et al. (2018).

Table 2: Models comparison. Mean scores and standard deviations over five training runs. Right most columns presents score for random agent and human.

Game	Ours, deterministic		Ours, det. recurrent		Ours, SD long		Ours, SD		Ours, SD $\gamma = 0.90$		Ours, SD $\gamma = 0.95$		Ours, SD 100 steps		Ours, SD 25 steps		random	human
Alien	378.3	(85.5)	321.7	(50.7)	**616.9**	(252.2)	405.2	(130.8)	413.0	(89.7)	590.2	(57.8)	435.6	(78.9)	534.8	(166.2)	184.8	7128.0
Amidar	62.4	(15.2)	86.7	(18.8)	74.3	(28.3)	**88.0**	(23.8)	50.3	(11.7)	78.3	(18.8)	37.7	(15.1)	82.2	(43.0)	11.8	1720.0
Assault	361.4	(166.6)	490.5	(143.6)	527.2	(112.3)	369.3	(107.8)	406.7	(118.7)	549.0	(127.9)	311.7	(88.2)	**664.5**	(298.2)	233.7	742.0
Asterix	668.0	(294.1)	**1853.0**	(391.8)	1128.3	(211.8)	1089.5	(335.3)	855.0	(176.4)	921.6	(114.2)	777.0	(200.4)	1340.6	(627.5)	248.8	8503.0
Asteroids	743.7	(92.2)	821.7	(115.6)	793.6	(182.2)	731.0	(165.3)	882.0	(24.7)	**886.8**	(45.2)	821.9	(93.8)	644.5	(110.6)	649.0	47389.0
Atlantis	14623.4	(2122.5)	12584.4	(5823.6)	**20992.5**	(11062.0)	14481.6	(2436.9)	18444.1	(4616.0)	14055.6	(6226.1)	14139.7	(2500.9)	11641.2	(3385.0)	16492.0	29028.0
BankHeist	13.8	(2.5)	15.1	(2.2)	**34.2**	(29.2)	8.2	(4.4)	11.9	(2.5)	12.0	(1.4)	13.1	(3.2)	12.7	(4.7)	15.0	753.0
BattleZone	3306.2	(794.1)	4665.6	(2799.4)	4031.2	(1156.1)	**5184.4**	(1347.5)	2781.2	(661.7)	4000.0	(788.9)	4068.8	(2912.1)	3746.9	(1426.8)	2895.0	37188.0
BeamRider	463.8	(29.2)	358.9	(87.4)	**621.6**	(79.8)	422.7	(103.6)	456.2	(160.8)	415.4	(103.4)	456.0	(60.9)	386.6	(264.4)	372.1	16926.0
Bowling	25.3	(10.4)	22.3	(17.0)	30.0	(5.8)	**34.4**	(16.3)	27.7	(5.2)	23.9	(3.3)	29.3	(7.5)	33.2	(15.5)	24.2	161.0
Boxing	-9.3	(10.9)	-3.1	(14.1)	7.8	(10.1)	9.1	(8.8)	**11.6**	(12.6)	5.1	(10.0)	-2.1	(5.0)	1.6	(14.7)	0.3	12.0
Breakout	6.1	(2.8)	10.2	(5.1)	**16.4**	(6.2)	12.7	(3.8)	7.3	(2.4)	8.8	(5.1)	11.4	(3.7)	7.8	(4.1)	0.9	30.0
ChopperCommand	906.9	(210.2)	709.1	(174.1)	979.4	(172.7)	**1246.9**	(392.0)	725.6	(204.2)	946.6	(49.9)	729.1	(185.1)	1047.2	(221.6)	671.0	7388.0
CrazyClimber	19380.0	(6138.8)	54700.3	(14480.5)	**62583.6**	(16856.8)	39827.8	(22582.6)	49840.9	(11920.9)	34353.1	(33547.2)	48651.2	(14903.5)	25612.2	(14037.5)	7339.5	35829.0
DemonAttack	191.9	(86.3)	120.3	(38.3)	**208.1**	(56.8)	169.5	(41.8)	187.5	(68.6)	194.9	(89.6)	170.1	(42.4)	202.2	(134.0)	140.0	1971.0
FishingDerby	-94.5	(3.0)	-96.9	(1.7)	-90.7	(5.3)	-91.5	(2.8)	-91.0	(4.1)	-92.6	(3.2)	**-90.0**	(2.7)	-94.5	(2.5)	-93.6	-39.0
Freeway	5.9	(13.1)	23.7	(13.5)	16.7	(15.7)	20.3	(18.5)	18.9	(17.2)	**27.7**	(13.3)	19.1	(16.7)	27.3	(5.8)	0.0	30.0
Frostbite	196.4	(4.4)	219.6	(21.4)	236.9	(31.5)	**254.7**	(4.9)	234.6	(26.8)	239.2	(19.1)	226.8	(16.9)	252.1	(54.4)	74.0	-
Gopher	510.2	(158.4)	225.2	(105.7)	596.8	(183.5)	771.0	(160.2)	**845.6**	(230.3)	612.6	(273.9)	698.4	(213.9)	509.7	(273.4)	245.9	2412.0
Gravitar	**237.0**	(73.1)	213.8	(57.4)	173.4	(54.7)	198.3	(39.9)	219.4	(7.8)	213.0	(37.3)	188.9	(27.6)	116.4	(84.0)	227.2	3351.0
Hero	621.5	(1281.3)	558.3	(1143.3)	2656.6	(483.1)	1295.1	(1600.1)	2853.9	(539.5)	**3503.5**	(892.9)	3052.7	(169.3)	1484.8	(1671.7)	224.6	30826.0
IceHockey	-12.6	(2.1)	-14.0	(1.8)	-11.6	(2.5)	**-10.5**	(2.2)	-12.2	(2.9)	-11.9	(1.2)	-13.5	(3.0)	-13.9	(3.9)	-9.7	1.0
Jamesbond	68.8	(37.2)	100.5	(69.8)	100.5	(36.8)	125.3	(112.5)	28.9	(12.7)	50.5	(21.3)	68.9	(42.7)	**163.4**	(81.8)	29.2	303.0
Kangaroo	**481.9**	(313.2)	191.9	(301.0)	51.2	(17.8)	323.1	(359.8)	148.1	(121.5)	37.5	(8.0)	301.2	(593.4)	340.0	(470.4)	42.0	3035.0
Krull	834.9	(166.3)	1778.5	(906.9)	2204.8	(776.5)	**4539.9**	(2470.4)	2396.5	(962.0)	2620.9	(856.2)	3559.0	(1896.7)	3320.6	(2410.1)	1543.3	2666.0
KungFuMaster	10340.9	(8835.7)	4086.6	(3384.5)	14862.5	(4031.6)	**17257.2**	(5502.6)	12587.8	(6810.0)	16926.6	(6598.3)	17121.2	(7211.6)	15541.2	(5086.1)	616.5	22736.0
MsPacman	560.6	(172.2)	1098.1	(450.9)	**1480.0**	(288.2)	762.8	(331.5)	1197.1	(544.6)	1273.3	(59.5)	921.0	(306.0)	805.8	(261.1)	235.2	6952.0
NameThisGame	1512.1	(408.3)	2007.9	(367.0)	**2420.7**	(289.4)	1990.4	(284.7)	2058.1	(103.7)	2114.8	(387.4)	2067.2	(304.8)	1805.3	(453.4)	2136.8	8049.0
Pong	-17.4	(5.2)	-11.6	(15.9)	**12.8**	(17.2)	5.2	(9.7)	-2.9	(7.3)	-2.5	(15.4)	-13.9	(7.7)	-1.0	(14.9)	-20.4	15.0
PrivateEye	16.4	(46.7)	50.8	(43.2)	35.0	(60.2)	58.3	(45.4)	54.4	(49.0)	67.8	(26.4)	88.3	(19.0)	**1334.3**	(1794.5)	26.6	69571.0
Qbert	480.4	(158.8)	603.7	(150.3)	**1288.8**	(1677.9)	559.8	(183.8)	899.3	(474.3)	1120.2	(697.1)	534.4	(162.5)	603.4	(138.2)	166.1	13455.0
Riverraid	1285.6	(604.6)	1740.7	(458.1)	1957.8	(758.1)	1587.0	(818.0)	1977.4	(332.7)	**2115.1**	(106.2)	1318.7	(540.4)	1426.0	(374.0)	1451.0	17118.0
RoadRunner	5724.4	(3093.1)	1228.8	(1025.9)	5640.6	(3936.6)	5169.4	(3939.0)	1586.2	(1574.1)	**8414.1**	(4542.8)	722.2	(627.2)	4366.2	(3867.8)	0.0	7845.0
Seaquest	419.5	(236.2)	289.6	(110.4)	**683.3**	(171.2)	370.9	(128.2)	364.6	(138.6)	337.8	(79.0)	247.8	(72.4)	350.0	(136.8)	61.1	42055.0
UpNDown	1329.3	(495.3)	926.7	(335.7)	**3350.3**	(3540.0)	2152.6	(1192.4)	1291.2	(324.6)	1250.6	(493.0)	1828.4	(688.3)	2136.5	(2095.0)	488.4	11693.0
YarsRevenge	3014.9	(397.4)	3291.4	(1097.3)	**5664.3**	(1870.5)	2980.2	(778.6)	2934.2	(459.2)	3366.6	(493.0)	2673.7	(216.8)	4666.1	(1889.4)	3121.2	54577.0

Table 3: Comparison of our method (SimPLe) with model-free benchmarks - PPO and Rainbow, trained with 100 thousands/500 thousands/1 million steps. (1 step equals 4 frames)

Game	SimPLe		PPO_100k		PPO_500k		PPO_1m		Rainbow_100k		Rainbow_500k		Rainbow_1m		random	human
Alien	616.9	(252.2)	291.0	(40.3)	269.0	(203.4)	362.0	(102.0)	290.6	(14.8)	828.6	(54.2)	945.0	(85.0)	184.8	7128.0
Amidar	74.3	(28.3)	56.5	(20.8)	93.2	(36.7)	123.8	(19.7)	20.8	(2.3)	194.0	(34.9)	275.8	(66.7)	11.8	1720.0
Assault	527.2	(112.3)	424.2	(55.8)	552.3	(110.4)	1134.4	(798.8)	300.3	(14.6)	1041.5	(92.1)	1581.8	(207.8)	233.7	742.0
Asterix	1128.3	(211.8)	385.0	(104.4)	1085.0	(354.8)	2185.0	(931.6)	285.7	(9.3)	1702.7	(162.8)	2151.6	(202.6)	248.8	8503.0
Asteroids	793.6	(182.2)	1134.0	(326.9)	1053.0	(433.3)	1251.0	(377.9)	912.3	(62.7)	895.9	(82.0)	1071.5	(91.7)	649.0	47389.0
Atlantis	20992.5	(11062.0)	34316.7	(5703.8)	4836416.7	(6218247.3)	-	(-)	17881.8	(617.6)	79541.0	(25393.4)	848800.0	(37533.1)	16492.0	29028.0
BankHeist	34.2	(29.2)	16.0	(12.4)	641.0	(352.8)	856.0	(376.7)	34.5	(2.0)	727.3	(198.3)	1053.3	(22.9)	15.0	753.0
BattleZone	4031.2	(1156.1)	5300.0	(3655.1)	14400.0	(6476.1)	19000.0	(4571.7)	3363.5	(523.8)	19507.1	(3193.3)	22391.4	(7708.9)	2895.0	37188.0
BeamRider	621.6	(79.8)	563.6	(189.4)	497.6	(103.5)	684.0	(168.8)	365.6	(29.8)	5890.0	(525.6)	6945.3	(1390.8)	372.1	16926.0
Bowling	30.0	(5.8)	17.7	(11.2)	28.5	(3.4)	35.8	(6.2)	24.7	(0.8)	31.0	(1.9)	30.6	(6.2)	24.2	161.0
Boxing	7.8	(10.1)	-3.9	(6.4)	3.5	(3.5)	19.6	(20.9)	0.9	(1.7)	58.2	(16.5)	80.3	(5.6)	0.3	12.0
Breakout	16.4	(6.2)	5.9	(3.3)	66.1	(114.3)	128.0	(153.3)	3.3	(0.1)	26.7	(2.4)	38.7	(3.4)	0.9	30.0
ChopperCommand	979.4	(172.7)	730.0	(199.0)	860.0	(285.3)	970.0	(201.5)	776.6	(59.0)	1765.2	(280.7)	2474.0	(504.5)	671.0	7388.0
CrazyClimber	62583.6	(16856.8)	18400.0	(5275.1)	33420.0	(3628.3)	58000.0	(16994.6)	12558.3	(674.6)	75655.1	(9439.6)	97088.1	(9975.4)	7339.5	35829.0
DemonAttack	208.1	(56.8)	192.5	(83.1)	216.5	(96.2)	241.0	(135.0)	431.6	(79.5)	3642.1	(478.2)	5478.6	(297.9)	140.0	1971.0
FishingDerby	-90.7	(5.3)	-95.6	(4.3)	-87.2	(5.3)	-88.8	(4.0)	-91.1	(2.1)	-66.7	(6.0)	-23.2	(22.3)	-93.6	-39.0
Freeway	16.7	(15.7)	8.0	(9.8)	14.0	(11.5)	20.8	(11.1)	0.1	(0.1)	12.6	(15.4)	13.0	(15.9)	0.0	30.0
Frostbite	236.9	(31.5)	174.0	(40.7)	214.0	(10.2)	229.0	(20.6)	140.1	(2.7)	1386.1	(321.7)	2972.3	(284.9)	74.0	-
Gopher	596.8	(183.5)	246.0	(103.3)	560.0	(118.8)	696.0	(279.3)	748.3	(105.4)	1640.5	(105.6)	1905.0	(211.1)	245.9	2412.0
Gravitar	173.4	(54.7)	235.0	(197.2)	235.0	(134.7)	325.0	(85.1)	231.4	(50.7)	214.9	(27.6)	260.0	(22.7)	227.2	3351.0
Hero	2656.6	(483.1)	569.0	(1100.9)	1824.0	(1461.2)	3719.0	(1306.0)	2676.3	(93.7)	10664.3	(1060.5)	13295.5	(261.2)	224.6	30826.0
IceHockey	-11.6	(2.5)	-10.0	(2.1)	-6.6	(1.6)	-5.3	(1.7)	-9.5	(0.8)	-9.7	(0.8)	-6.5	(0.5)	-9.7	1.0
Jamesbond	100.5	(36.8)	65.0	(46.4)	255.0	(101.7)	310.0	(129.0)	61.7	(8.8)	429.7	(27.9)	692.6	(316.2)	29.2	303.0
Kangaroo	51.2	(17.8)	140.0	(102.0)	340.0	(407.9)	840.0	(806.5)	38.7	(9.3)	970.9	(501.9)	4084.6	(1954.1)	42.0	3035.0
Krull	2204.8	(776.5)	3750.4	(3071.9)	3056.1	(1155.5)	5061.8	(1333.4)	2978.8	(148.4)	4139.4	(336.2)	4971.1	(360.3)	1543.3	2666.0
KungFuMaster	14862.5	(4031.6)	4820.0	(983.2)	17370.0	(10707.6)	13780.0	(3971.6)	1019.4	(149.6)	19346.1	(3274.4)	21258.6	(3210.2)	616.5	22736.0
MsPacman	1480.0	(288.2)	496.0	(379.8)	306.0	(70.2)	594.0	(247.9)	364.3	(20.4)	1558.0	(248.9)	1881.4	(112.0)	235.2	6952.0
NameThisGame	2420.7	(289.4)	2225.0	(423.7)	2106.0	(898.8)	2311.0	(547.6)	2368.2	(318.3)	4886.5	(583.1)	4454.2	(338.3)	2136.8	8049.0
Pong	12.8	(17.2)	-20.5	(0.6)	-8.6	(14.9)	14.7	(5.1)	-19.5	(0.2)	19.9	(0.4)	20.6	(0.2)	-20.4	15.0
PrivateEye	35.0	(60.2)	10.0	(20.0)	20.0	(40.0)	20.0	(40.0)	42.1	(53.8)	-6.2	(89.8)	2336.7	(4732.6)	26.6	69571.0
Qbert	1288.8	(1677.9)	362.5	(117.8)	757.5	(78.9)	2675.0	(1701.1)	235.6	(12.9)	4241.7	(193.1)	8885.2	(1690.9)	166.1	13455.0
Riverraid	1957.8	(758.1)	1398.0	(513.8)	2865.0	(327.1)	2887.0	(807.0)	1904.2	(44.2)	5068.6	(292.6)	7018.9	(334.2)	1451.0	17118.0
RoadRunner	5640.6	(3936.6)	1430.0	(760.0)	5750.0	(5259.9)	8930.0	(4304.0)	524.1	(147.5)	18415.4	(5280.0)	31379.7	(3225.8)	0.0	7845.0
Seaquest	683.3	(171.2)	370.0	(103.3)	692.0	(48.3)	882.0	(122.7)	206.3	(17.1)	1558.7	(221.2)	3279.9	(683.9)	61.1	42055.0
UpNDown	3350.3	(3540.0)	2874.0	(1105.8)	12126.0	(1389.5)	13777.0	(6766.3)	1346.3	(95.1)	6120.7	(356.8)	8010.9	(907.0)	488.4	11693.0
YarsRevenge	5664.3	(1870.5)	5182.0	(1209.3)	8064.8	(2859.8)	9495.0	(2638.3)	3649.0	(168.6)	7005.7	(394.2)	8225.1	(957.9)	3121.2	54577.0

Table 4: Models comparison. Scores of median (left) and best (right) models out of five training runs. Right most columns presents score for random agent and human.

Game	Ours, deterministic		Ours, det. recurrent		Ours, SD long		Ours, SD		Ours, SD $\gamma = 0.90$		Ours, SD $\gamma = 0.95$		SD 100 steps		Ours, SD 25 steps		random	human
Alien	354.4	516.6	299.2	381.1	515.9	1030.5	409.2	586.9	411.9	530.5	567.3	682.7	399.5	522.3	525.5	792.8	184.8	7128.0
Amidar	58.0	84.8	82.7	118.4	80.2	102.7	85.1	114.0	55.1	58.9	84.3	101.4	45.2	47.5	93.1	137.7	11.8	1720.0
Assault	334.4	560.1	566.6	627.2	509.1	671.1	355.7	527.9	369.1	614.4	508.4	722.5	322.9	391.1	701.4	1060.3	233.7	742.0
Asterix	529.7	1087.5	1798.4	2282.0	1065.6	1485.2	1158.6	1393.8	805.5	1159.4	923.4	1034.4	813.3	1000.0	1128.1	2313.3	248.8	8503.0
Asteroids	727.3	854.7	827.7	919.8	899.7	955.6	671.2	962.0	885.5	909.1	886.1	949.5	813.8	962.2	657.5	752.7	649.0	47389.0
Atlantis	15587.5	16545.3	15939.1	17778.1	13695.3	34890.6	13645.3	18396.9	19367.2	23046.9	12981.2	23579.7	15020.3	16790.6	12196.9	15728.1	16492.0	29028.0
BankHeist	14.4	16.2	14.7	18.8	31.9	77.5	8.9	13.9	12.3	14.5	12.3	13.1	12.8	17.2	14.1	17.0	15.0	753.0
BattleZone	3312.5	4140.6	4515.6	9312.5	3484.4	5359.4	5390.6	7093.8	2937.5	3343.8	4421.9	4703.1	3500.0	8906.2	3859.4	5734.4	2895.0	37188.0
BeamRider	453.1	515.5	351.4	470.2	580.2	728.8	433.9	512.6	393.5	682.8	446.6	519.2	447.1	544.6	385.7	741.9	372.1	16926.0
Bowling	27.0	36.2	28.4	43.7	28.0	39.6	24.9	55.0	27.7	34.9	22.6	28.6	28.4	39.9	37.0	54.7	24.2	161.0
Boxing	-7.1	0.2	3.5	5.0	9.4	21.0	8.3	21.5	6.4	31.5	2.5	15.0	-0.7	2.2	-0.9	20.8	0.3	12.0
Breakout	5.5	9.8	12.5	13.9	16.0	22.8	11.0	19.5	7.4	10.4	10.2	14.1	10.5	16.7	6.9	13.0	0.9	30.0
ChopperCommand	942.2	1167.2	748.4	957.8	909.4	1279.7	1139.1	1909.4	682.8	1045.3	954.7	1010.9	751.6	989.1	1031.2	1329.7	671.0	7388.0
CrazyClimber	20754.7	23831.2	49854.7	80156.2	55795.3	87593.8	41396.9	67250.0	56875.0	58979.7	19448.4	84070.3	53406.2	64196.9	19345.3	43179.7	7339.5	35829.0
DemonAttack	219.2	263.0	135.8	148.4	191.2	288.9	182.4	223.9	160.3	293.8	204.1	312.8	164.4	222.6	187.5	424.8	140.0	1971.0
FishingDerby	-94.3	-90.2	-97.3	-94.2	-91.8	-84.3	-91.6	-88.6	-90.0	-85.7	-92.0	-88.8	-90.6	-85.4	-95.0	-90.7	-93.6	-39.0
Freeway	0.0	29.3	29.3	32.2	21.5	32.0	33.5	34.0	31.1	32.0	33.5	33.8	30.0	32.3	29.9	33.5	0.0	30.0
Frostbite	194.5	203.9	213.4	256.2	248.8	266.9	253.1	262.8	246.7	261.7	250.0	255.9	215.8	247.7	249.4	337.5	74.0	-
Gopher	514.7	740.6	270.3	320.9	525.3	845.6	856.9	934.4	874.1	1167.2	604.1	1001.6	726.9	891.6	526.2	845.0	245.9	2412.0
Gravitar	232.8	310.2	219.5	300.0	156.2	233.6	202.3	252.3	223.4	225.8	228.1	243.8	193.8	218.0	93.0	240.6	227.2	3351.0
Hero	71.5	2913.0	75.0	2601.5	2935.0	3061.6	237.5	3133.8	3135.0	3147.5	3066.2	5092.0	3067.3	3256.9	1487.2	2964.8	224.6	30826.0
IceHockey	-12.4	-9.9	-14.8	-11.8	-12.3	-7.2	-10.0	-7.7	-11.8	-8.5	-11.6	-10.7	-12.9	-10.0	-12.2	-11.0	-9.7	1.0
Jamesbond	64.8	128.9	64.8	219.5	110.9	141.4	87.5	323.4	25.0	46.9	58.6	69.5	61.7	139.1	139.8	261.7	29.2	303.0
Kangaroo	500.0	828.1	68.8	728.1	62.5	65.6	215.6	909.4	103.1	334.4	34.4	50.0	43.8	1362.5	56.2	1128.1	42.0	3035.0
Krull	852.2	1014.3	1783.6	2943.6	1933.7	3317.5	4264.3	7163.2	1874.8	3554.5	2254.0	3827.1	3142.8	6315.2	3198.2	6833.4	1543.3	2666.0
KungFuMaster	7575.0	20450.0	4848.4	8065.6	14318.8	21054.7	17448.4	21943.8	12964.1	21956.2	20195.3	23690.6	19718.8	25375.0	18025.0	20365.6	616.5	22736.0
MsPacman	557.3	818.0	1178.8	1685.9	1525.0	1903.4	751.2	1146.1	1410.5	1538.9	1277.3	1354.5	866.2	1401.9	777.2	1227.8	235.2	6952.0
NameThisGame	1468.1	1992.7	1826.7	2614.5	2460.0	2782.8	1919.8	2377.7	2087.3	2155.2	1994.8	2570.3	2153.4	2471.9	1964.2	2314.8	2136.8	8049.0
Pong	-19.6	-8.5	-17.3	16.7	20.7	21.0	1.4	21.0	-2.0	6.6	3.8	14.2	-17.9	-2.0	-10.1	21.0	-20.4	15.0
PrivateEye	0.0	98.9	75.0	82.8	0.0	100.0	76.6	100.0	75.0	96.9	60.9	100.0	96.9	99.3	100.0	4038.7	26.6	69571.0
Qbert	476.6	702.7	555.9	869.9	656.2	4259.0	508.6	802.7	802.3	1721.9	974.6	2322.3	475.0	812.5	668.8	747.3	166.1	13455.0
Riverraid	1416.1	1929.4	1784.4	2274.5	2360.0	2659.8	1799.4	2158.4	2053.8	2307.5	2143.6	2221.2	1387.8	1759.8	1345.5	1923.4	1451.0	17118.0
RoadRunner	5901.6	8484.4	781.2	2857.8	5906.2	11176.6	2804.7	10676.6	1620.3	4104.7	7032.8	14978.1	857.8	1342.2	2717.2	8560.9	0.0	7845.0
Seaquest	414.4	768.1	236.9	470.6	711.6	854.1	386.9	497.2	330.9	551.2	332.8	460.9	274.1	317.2	366.9	527.2	61.1	42055.0
UpNDown	1195.9	2071.1	1007.5	1315.2	1616.1	8614.5	2389.5	3798.3	1433.3	1622.0	1248.6	1999.4	1670.3	2728.0	1825.2	5193.1	488.4	11693.0
YarsRevenge	3047.0	3380.5	3416.3	4230.8	6580.2	7547.4	2435.5	3914.1	2955.9	3314.5	3434.8	3896.3	2745.3	2848.1	4276.3	6673.1	3121.2	54577.0

E BASELINES OPTIMIZATION

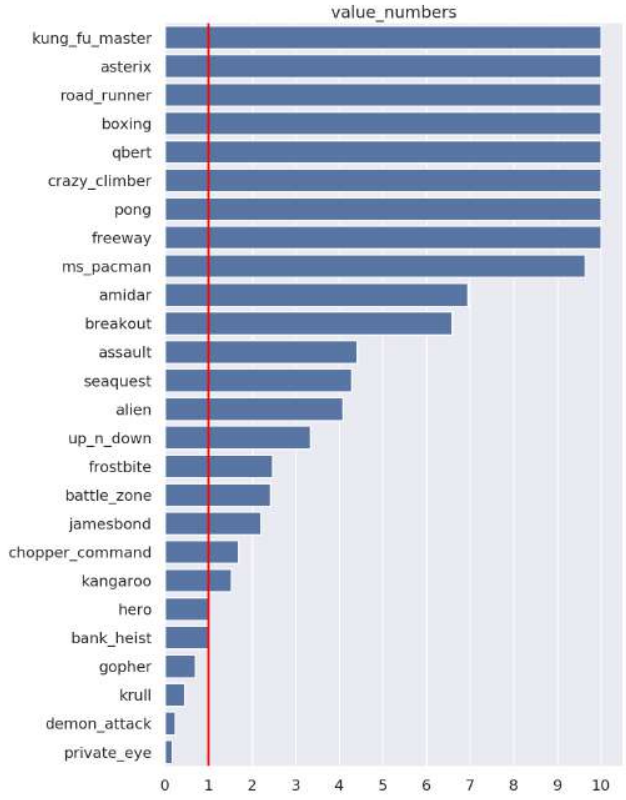
To assess the performance of SimPle we compare it with model-free algorithms. To make this comparison more reliable we tuned Rainbow in the low data regime. To this end we run an hyperparameter search over the following parameters from `https://github.com/google/dopamine/blob/master/dopamine/agents/rainbow/rainbow_agent.py`:

- `update_horizon` in {1, 3}, best parameter = 3
- `min_replay_history` in {500, 5000, 20000}, best parameter = 20000
- `update_period` in {1, 4}, best parameter = 4
- `target_update_period` {50, 100, 1000, 4000}, best parameter = 8000
- `replay_scheme` in {`uniform`, `prioritized`}, best parameter = `prioritized`

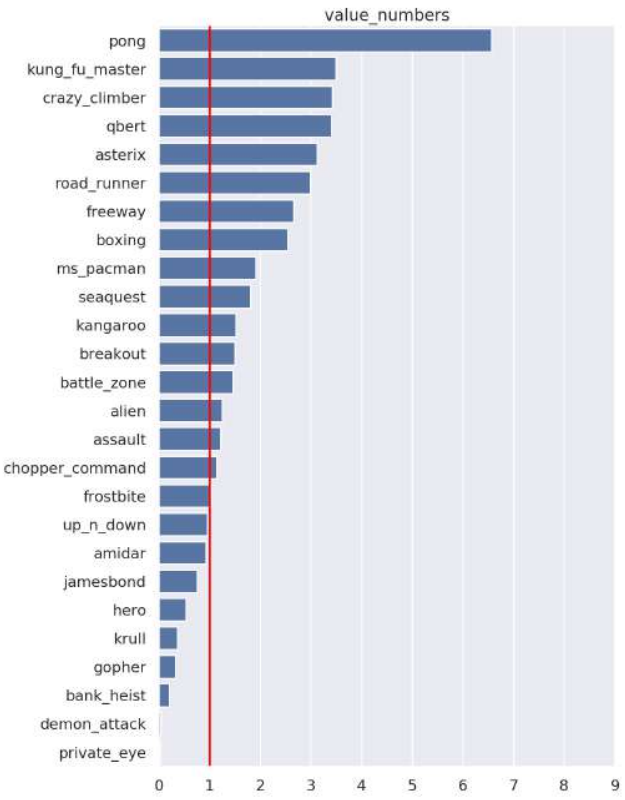
Each set of hyperparameters was used to train 5 Rainbow agents on the game of `Pong` until 1 million of interactions with the environment. Their average performance was used to pick the best hyperparameter set.

For PPO we used the standard set of hyperparameters from `https://github.com/openai/baselines`.

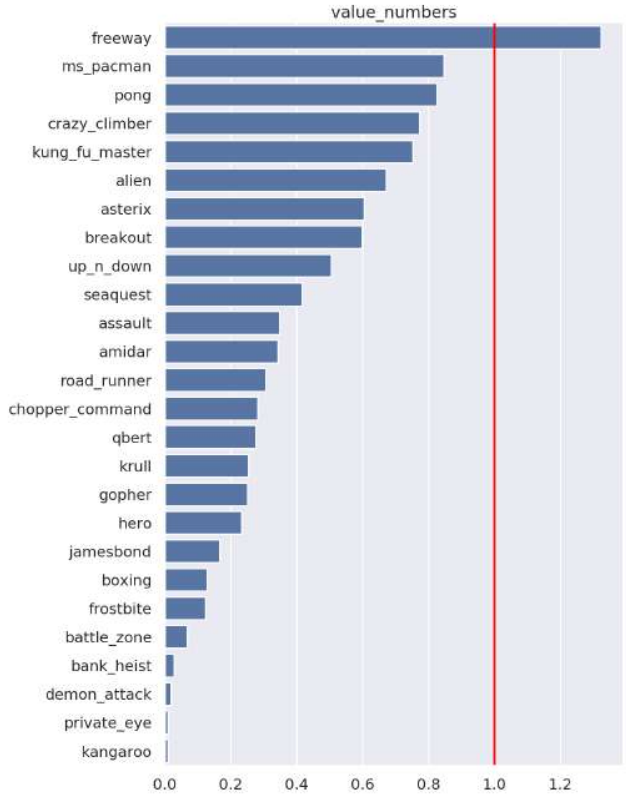
F RESULTS AT DIFFERENT NUMBERS OF INTERACTIONS



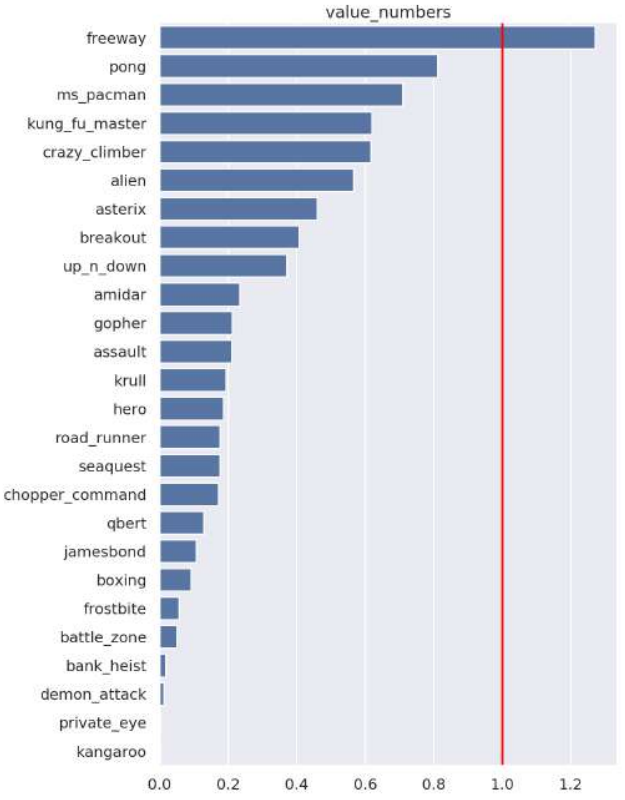
(a) Fraction at 100K clipped to 10.



(b) Fraction at 200K

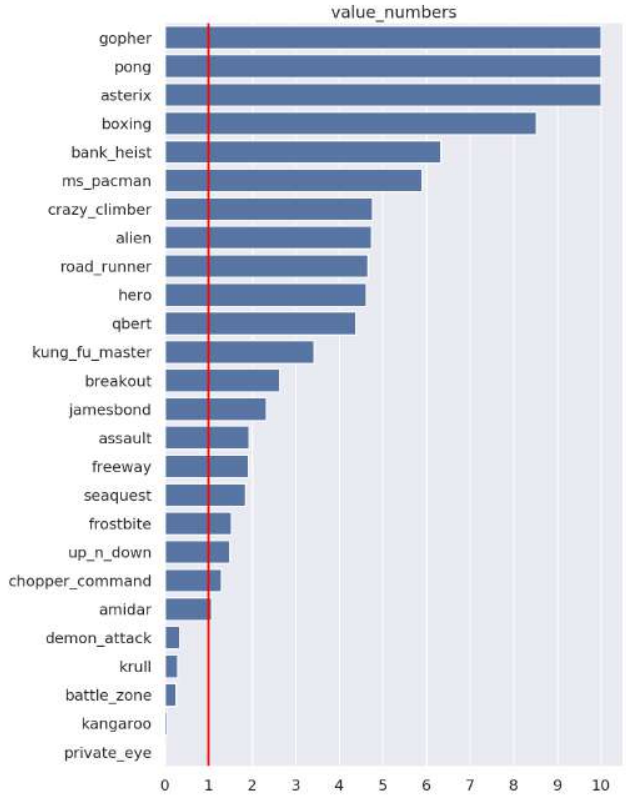


(c) Fraction at 500K.

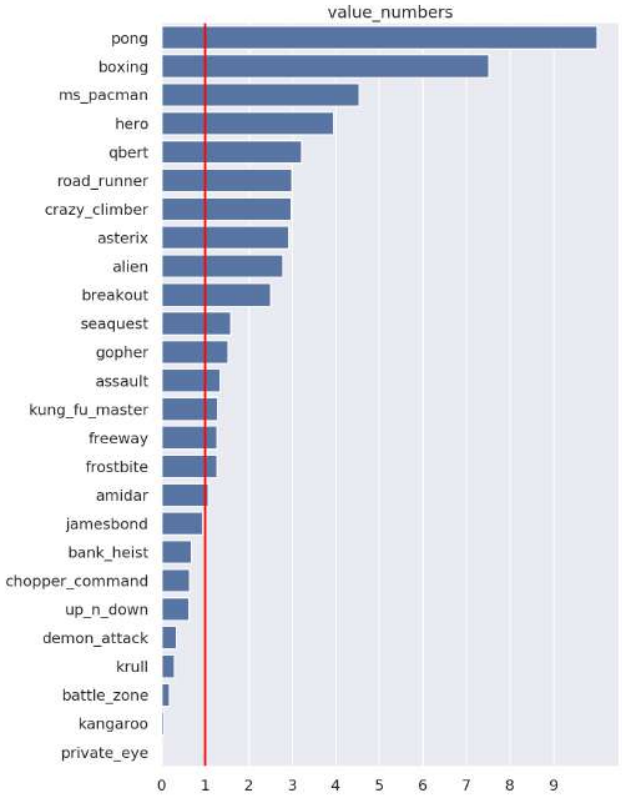


(d) Fraction at 1M.

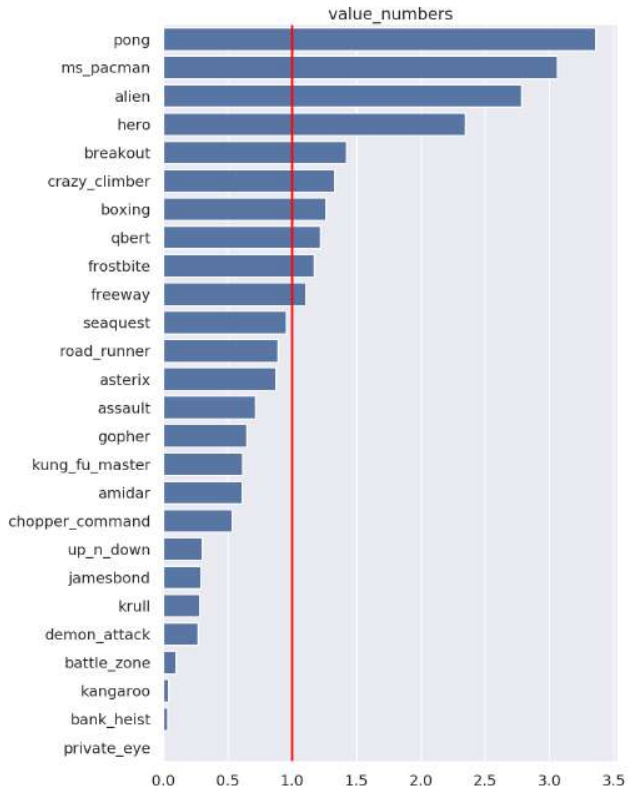
Figure 12: Fractions of the rainbow scores at given number of samples. These were calculate with the formula $(SimPLe\_score - random\_score)/(rainbow\_score - random\_score)$*; if denominator is smaller than 0, both nominator and denominator are increased by 1.*



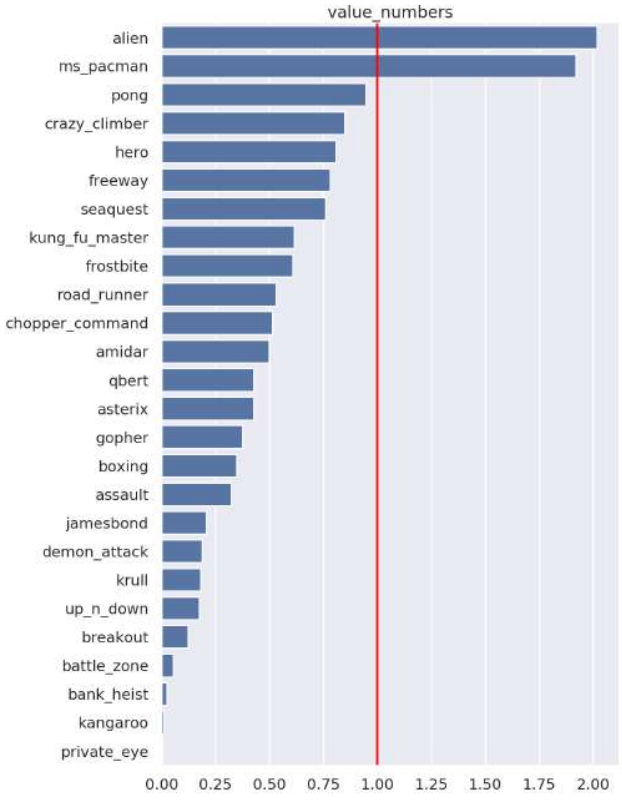
(a) Fraction at 100K clipped to 10.



(b) Fraction at 200K

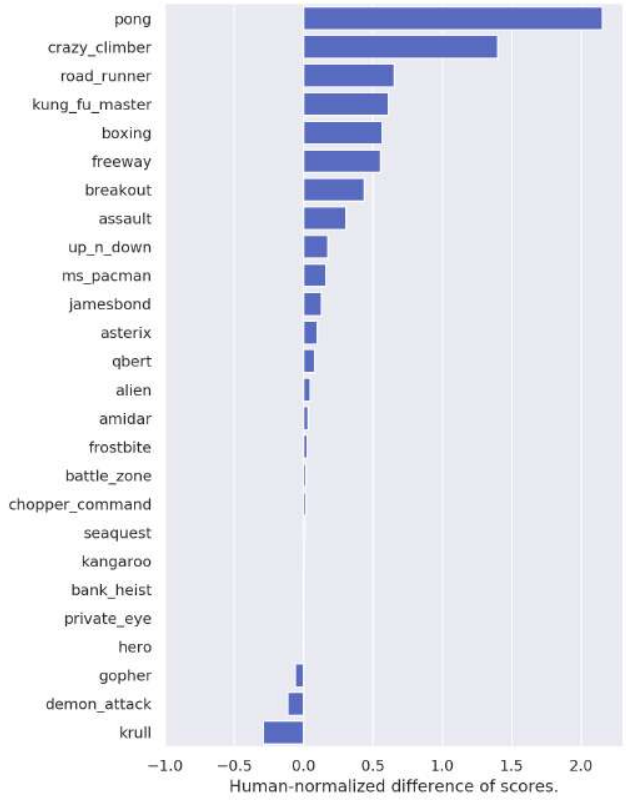


(c) Fraction at 500K.

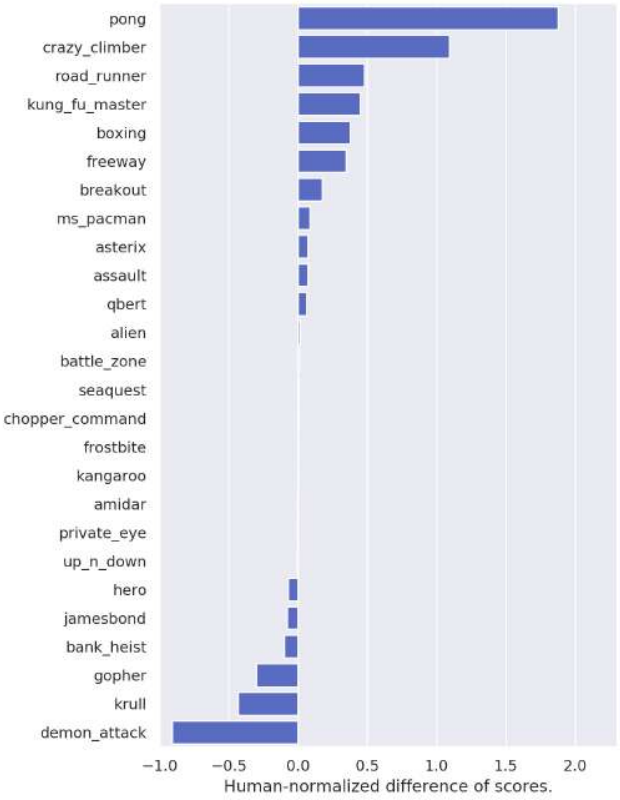


(d) Fraction at 1M.

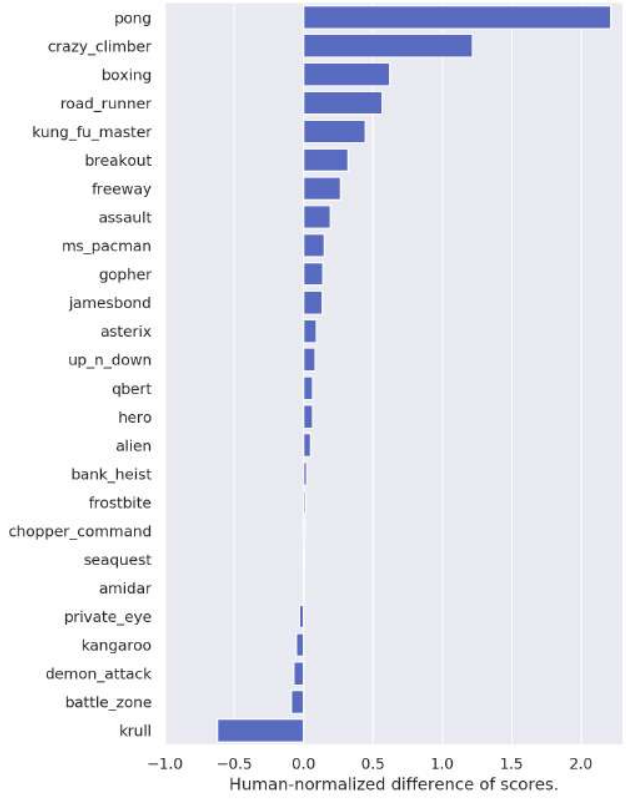
Figure 13: Fractions of the ppo scores at given number of samples. These were calculate with the formula $(SimPLe\_score - random\_score)/(ppo\_score - random\_score)$*; if denominator is smaller than 0, both nominator and denominator are increased by 1.*



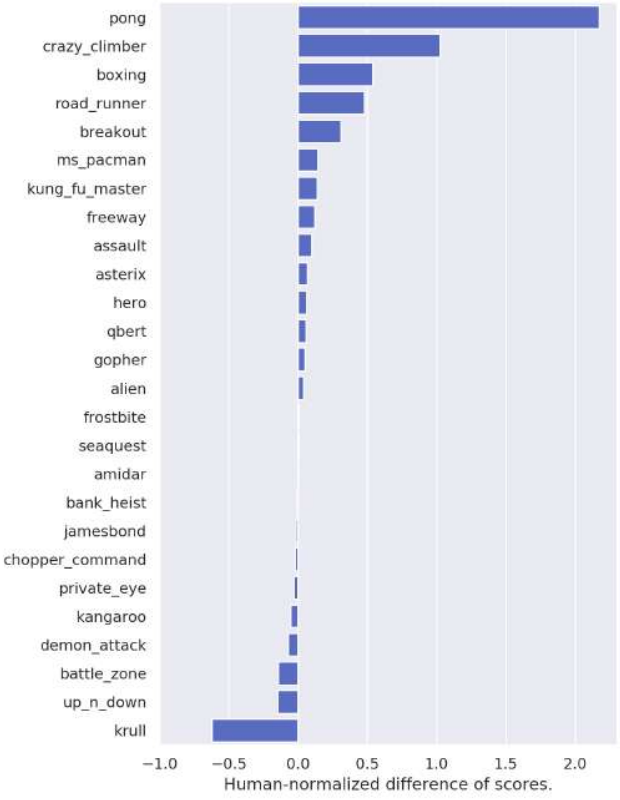
(a) SimPLe compared to Rainbow at 100K.



(b) SimPLe compared to Rainbow at 200K



(c) SimPLe compared to PPO at 100K.



(d) SimPLe compared to PPO at 200K.

Figure 14: Comparison of scores from Simple against Rainbow and PPO at different numbers of interactions. The following formula is used: $(SimPLe\_score@100K - baseline\_score)/human\_score$. Points are normalized by average human score in order to be presentable in one graph.

2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Simulation-Based Reinforcement Learning for Real-World Autonomous Driving

Błażej Osiński[1,3], Adam Jakubowski[1], Paweł Zięcina[1], Piotr Miłoś[1,5], Christopher Galias[1,4], Silviu Homoceanu[2] and Henryk Michalewski[3]

***Abstract*— We use reinforcement learning in simulation to obtain a driving system controlling a full-size real-world vehicle. The driving policy takes RGB images from a single camera and their semantic segmentation as input. We use mostly synthetic data, with labelled real-world data appearing only in the training of the segmentation network.**

Using reinforcement learning in simulation and synthetic data is motivated by lowering costs and engineering effort.

In real-world experiments we confirm that we achieved successful sim-to-real policy transfer. Based on the extensive evaluation, we analyze how design decisions about perception, control, and training impact the real-world performance.

I. Introduction

This work focuses on verifying whether it is possible to obtain a driving system using data from a simulator, which can be deployed on a real car. Using synthetic data, in comparison to collecting it in the real world, reduces the cost of developing such a system. Our policies were trained using reinforcement learning (RL) and confirmed to be useful in driving a real, full-sized passenger vehicle with state-of-the-art equipment required for Level 4 autonomy. The real-world tests consisted of 9 driving scenarios of total length of about 2.5 km.

The driving policy is evaluated by its real-world performance on multiple scenarios outlined in Section III–a. To complete a scenario, the driving agent needs to execute from 250 to 700 actions at 10 Hz at speeds varying from 15 to 30 km/h. In some of our experiments, the learned controller outputs the steering command directly. In other, the controller outputs waypoints which are transformed to steering commands using a proprietary control system. In this work, we decided to limit intermediate human-designed or learned representations of the real world only to semantic segmentation. The semantic segmentator used in our system is the only component trained using the real-world data – its training process mixes real-world and synthetic images. Our driving policies are trained only in simulation and directly on visual inputs, understood as RGB images along with their segmentation masks. The input contains also selected car metrics and a high-level navigation command inspired by [8].

Using reinforcement learning and RGB inputs was a deliberate design decision. The goal behind this choice was to answer the following research questions: Is it possible to train a driving policy in an end-to-end fashion? Is such a policy obtained in simulation capable of real-world driving?

Using simulation and reinforcement learning is instrumental in generating rich experience. The former enables avoiding hard safety constraints of real-world scenarios and lets the latter to explore beyond the imagination of hand-designed situations. Furthermore, RL enables end-to-end training with none or little human intervention. This property is highly desirable as it greatly reduces the engineering effort. It may also eliminate errors arising when gluing a heterogeneous system consisting of separate perception and control modules.

The quality of a simulator is crucial to obtain transferable polices. We use CARLA [13], which offers reasonable fidelity of physical and visual layers. This fidelity comes at a high computation cost, however. In our case, it was alleviated by implementing a parallelized training architecture inspired by IMPALA [14]. With our current infrastructure, we gathered as much as 100 years of simulated driving experience. This enabled us to conduct several lines of experiments testing various design choices. These experiments constitute the main contribution of this work:

1. In simulation: we verify the influence of visual randomizations on transfer between different scenarios in simulation; results are summarized in Section IV-A.

2. In the real world: we test 10 models listed in Table I on 9 driving scenarios. In total we report results gathered over more than 400 test drives. See Section IV-B for a detailed description.

model	description
`CONTINUOUS-PLAIN`	base experiment
`CONTINUOUS-LOW-RAND`	experiment using smaller number of randomizations
`DISCRETE-PLAIN`	model using discrete actions
`CONTINUOUS-REG`	experiment with additional l_2 regularization
`DISCRETE-REG`	analog of `DISCRETE-PLAIN` with additional l_2 regularization
`SEMSEG-ONLY`	model with semantic segmentation as only visual input
`WAYPOINTS-DISCRETE`	model driving with waypoints
`AUXILIARY-DEPTH`	model predicting depth as auxiliary task
`DYNAMICS-RAND-FFW`	feed-forward model trained with dynamics randomizations
`DYNAMICS-RAND-RNN`	model with memory trained with dynamics randomizations

TABLE I: Summary of models evaluated in this work.

In Section IV-C we describe two failure cases and in Section IV-D we assess a proxy metric potentially useful for offline evaluation of models. We provide recordings from 9 autonomous test drives at `https://bit.ly/2k8syvh`. The test drives correspond to the scenarios listed in Figures 1.

II. Related work

a) Synthetic data and real-world robotics: Synthetic images were used in the ALVINN experiment [30]. [35]

[1] deepsense.ai, Warsaw, Poland
[2] Volkswagen AG, Wolfsburg, Germany
[3] University of Warsaw, Warsaw, Poland
[4] Jagiellonian University, Cracow, Poland
[5] Institute of Mathematics of the Polish Academy of Sciences, Warsaw, Poland

978-1-7281-7395-5/20/$31.00 ©2020 IEEE

proposed a training procedure for drones and [18], [29], [28], [44], [27] proposed experiments with robotic manipulators where training was performed using only synthetic data. Progressive nets and data generated using simulator were used in [33] to learn policies in the domain of real-world robot manipulation. A driving policy for a one-person vehicle was trained in [2]. The policy in [2] is reported to show good performance on a rural road and the training used mostly synthetic data generated by Unreal Engine 4. Our inclusion of segmentation as described in Section III–d is inspired by sim-to-real experiments presented in [19] and [25]. Visual steering systems inspired by [35] and trained using synthetic data were presented in [36], [34].

b) Synthetic data and simulated robotics: Emergence of high-quality general purpose physics engines such as MuJoCo [45], along with game engines such as Unreal Engine 4 and Unity, and their specialized extensions such as CARLA [13] or AirSim [40], allowed for creation of sophisticated photo-realistic environments which can be used for training and testing of autonomous vehicles. A deep RL framework for autonomous driving was proposed in [37] and tested using the racing car simulator TORCS. Reinforcement learning methods led to very good performance in simulated robotics – see, for example, solutions to complicated walking tasks in [16], [23]. In the context of CARLA, impressive driving policies were trained using imitation learning [8], [31], affordance learning [38], reinforcement learning [4], and a combination of model-based and imitation learning methods proposed in [31]. However, as stated in [2]: "training and evaluating methods purely in simulation is often 'doomed to succeed' at the desired task in a simulated environment" and indeed, in our suite of experiments described in Section III most of the simulated tasks can be relatively easily solved, in particular when a given environment is deterministic and simulated observations are not perturbed.

c) Reinforcement learning and real-world robotics: A survey of various applications of RL in robotics can be found in [9, Section 2.5]. The role of simulators and RL in robotics is discussed in [41, Section IV]. In [35], [29], [28], [44], [27], [19], [2], [33] policies are deployed on real-world robots and training is performed mostly using data generated by simulators. [21] proposes a system with dynamics trained using real-world data and perception trained using synthetic data. Training of an RL policy in the TORCS engine with a real-world deployment is presented in [42].

III. Environment and learning algorithm

We use CARLA [13], an open-source simulator for autonomous driving research based on Unreal Engine 4. CARLA features open assets, including seven built-in maps, 14 predefined weather settings, and multiple vehicles with different physical parameters. In our experiments we use input from simulated cameras; their settings, including position, orientation, and field of view, can be customized. Two visual quality levels (`LOW` and `EPIC`) are supported; the latter implements visual features including shadows, water reflections, sun flare effects, and antialiasing.

Below we describe our experimental setup as used in the basic `CONTINUOUS-PLAIN` experiment. We varied its various elements in other experiments. Details are provided in Section IV-B.

a) Simulated and real-world scenarios: The models were tested on 9 real-world scenarios presented in Figure 1. These scenarios contain diverse driving situations, including turns and the entry and exit of an overpass. For training, we developed new CARLA-compatible maps resembling the real-world testing area (approximately 50% of the testing scenarios were covered). We used these maps along with maps provided in CARLA for training, with some scenarios reserved for validation. In all scenarios the agent's goal is to follow a route from start to finish. These routes are lists of checkpoints: they are generated procedurally in CARLA maps and predefined in maps developed by us.

In training, agents are expected to drive in their own lanes, but other traffic rules are ignored. Moreover, we assume that the simulated environment is static, without any moving cars or pedestrians, hence a number of a safety driver interventions during test deployments in real traffic is unavoidable.

b) Rewards in simulation and metrics of real-world performance: In simulation, the agent is rewarded for following a reference trajectory, which provides a dense training signal. The episode fails if the agent diverges from the trajectory more than 5 meters or collides with an obstacle. In the real world, for each scenario, we measure the percentage of distance driven autonomously (i.e. without human intervention); results are presented in Figure 3. Since tests were made in an uncontrolled environment with other vehicles and pedestrians, the safety driver was instructed to take over in all situations which were potentially risky. We also measure divergence from expert trajectories (see Figure 4).

c) Actions: Vehicles are controlled by two values: throttle and steering. The throttle is controlled by a PID controller with speed set to a constant, and thus our neural network policies only control the steering. We explore various possibilities for action spaces. Unless stated otherwise, in training the policy is modeled as a Gaussian distribution over the angle of the steering wheel. In evaluations we use the mean of the distribution.

d) Semantic segmentation: The semantic segmentation model is trained in a supervised way separately from the reinforcement learning loop. We used the U-Net [32] architecture and synthetic data from CARLA (which can render both RGB images and their ground-truth semantic labels), the Mapillary dataset [26], as well as real-world labeled data from an environment similar to the one used in test drives. The output of the model is further simplified to include only the classes most relevant to our problem: road, road marking, and everything else (i.e. obstacles).

e) Observations: The observation provided to the agent consists of visual input and two car metrics (speed and acceleration). In order to disambiguate certain road situations (e.g. intersections), a high-level navigation command is also given: `lane follow`, `turn right/left` or `go straight`. The commands resemble what could be provided

Fig. 1: All real-world scenarios used in our experiments. Left map: (a) `autouni-arc`, (b) `autouni-straight`. Center map: (c) `factory_city-overpass*`, (d) `factory_city-overpass_exit`. Right map: (e) `factory_city-tunnel-bt10*`, (f) `factory_city-bt10-u_turn`, (g) `factory_city-u_turn-sud_strasse`, (h) `factory_city-sud_strasse_u_turn*`, (i) `factory_city-u_turn-bt10*`. Scenarios marked with * were used for training in simulation.

by a typical GPS-based navigation system.

The visual input is based on an RGB image from a single front camera which is downscaled to 134×84 pixels. The camera position and orientation in simulation was configured to reflect the real-world setup. The RGB observation is concatenated with its semantic segmentation as described in the previous subsection. Including this component was motivated by [25], which claims that it "contains sufficient information for following the road and taking turns, but it is abstract enough to support transfer". We have further evaluated this claim in experiment `SEMSEG-ONLY`.

f) Domain randomizations: Randomizations are considered to be pivotal to achieve sim-to-real transfer (and robust polices in general; see e.g. [27]). In our experiments we used the following list of visual randomizations: 10 weather settings (we used CARLA weather presets, which affect only the visual features of the environment), the simulation quality (we used both LOW and EPIC), camera input randomizations (we used a set of visual augmentations, such as adding gaussian noise, varying brightness, and applying blur or cutout [10]). We recall that our policies are trained on multiple scenarios and different maps, which is also aimed to increase robustness. Unless stated otherwise these randomizations are used in all experiments. In a separate experiment we also evaluated using randomization of dynamics, see description in Section IV-B-e.

g) Network architecture: The RL policy is implemented using a neural network; see its simplified architecture in Figure 2. As the feature extractor for visual input (RGB and semantic segmentation) we use the network from [14]. Our choice was influenced by [6], where this network was shown to generalize well to different RL environments. Note that policy transfer between simulation and reality can be seen as a generalization challenge.

h) Learning algorithm: We used OpenAI Baselines [11] `ppo2` (see website `http://bit.ly/34xh7z4` for training hyperparameters). We typically use 4 simultaneous PPO trainers, which share gradients via Horovod [39]. Each

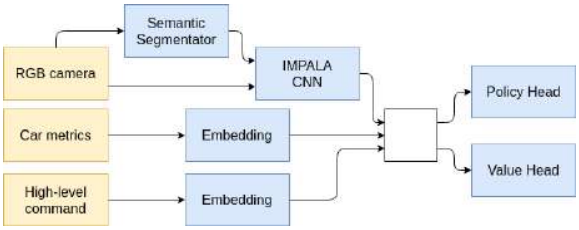


Fig. 2: Network architecture.

of them uses 2 Tesla K40 GPUs; one for running 10 Carla instances and the other for the optimisation of the policy network. In practice, this setup is able to gather $1.2 \cdot 10^7$ frames a day (equivalent to 13 days of driving).

Thanks to dense rewards the training in simulation was quite stable across models and hyperparameters. For deployments we have decided to use 1-4 models per experiment type, each trained using roughly 10^8 frames (equivalent to about 115 days of driving).

i) System identification: Inspired by the importance of system identification for sim-to-real transfer demonstrated in [43], we configured the CARLA simulator to mimic some values measured in the car used for deployment. These were the maximal steering angle and the time for a steering command to take effect (i.e. its delay).

IV. Experiments

Our models have been evaluated both in simulation and in reality, with much more focus on the latter. Below we provide detailed description of experiments conducted in both domains.

A. Experiment in simulation

In this experiment, we measure in simulation how randomizations affect performance. To this end, we apply fewer randomizations then the set used throughout all other experiments. Precisely, we used only one weather setting, trained only on the LOW quality settings and did not augment

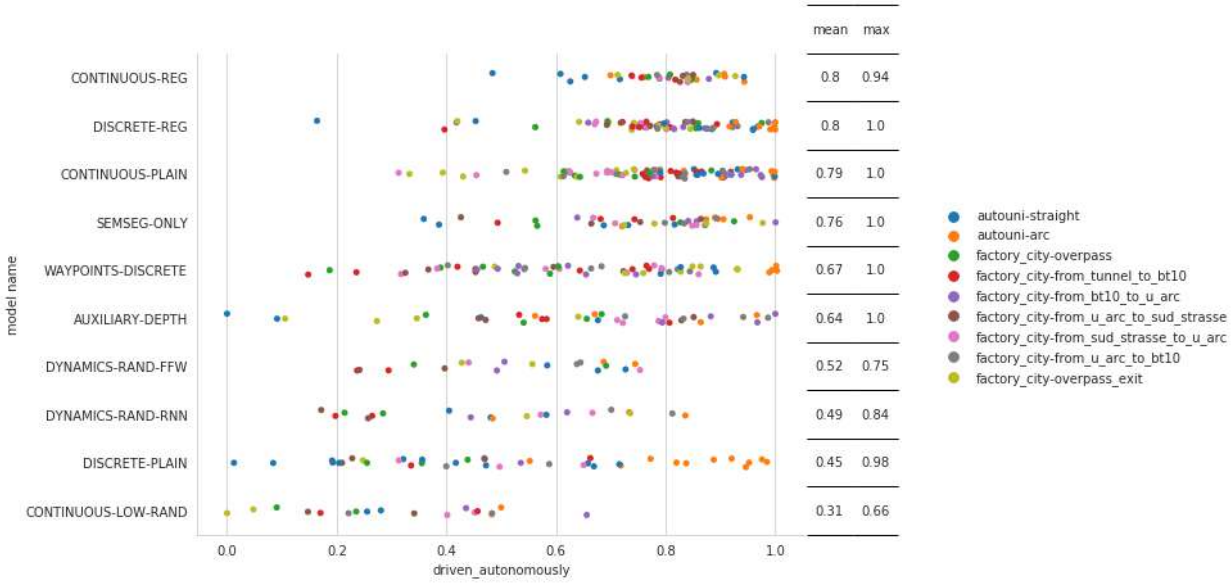


Fig. 3: Summary of experiments with baselines across nine real-world scenarios. The columns to the right show the mean and max of autonomy (the percentage of distance driven autonomously). Models are sorted according to their mean performance. Print in color for better readability.

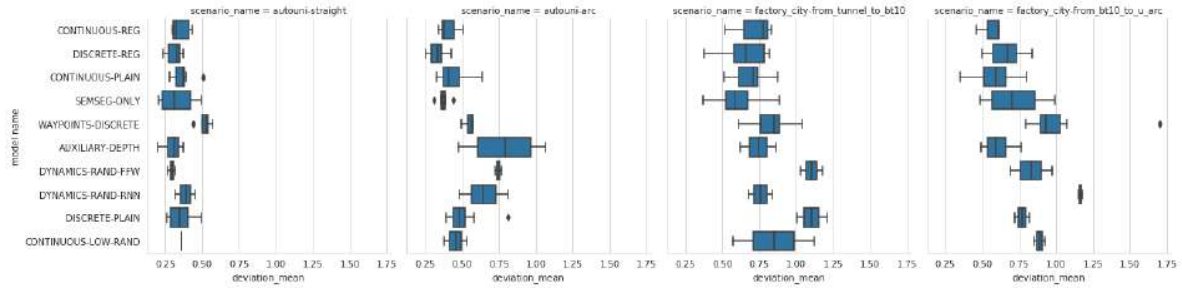


Fig. 4: Average deviation of models from expert trajectories. Measurements based on GPS. The graphs for all scenarios can be found on the website `http://bit.ly/34xh7z4`

the camera inputs. We conclude that the model trained with the standard set of randomizations generalizes better to the holdout town and with the holdout weather setting (see Figure 5).

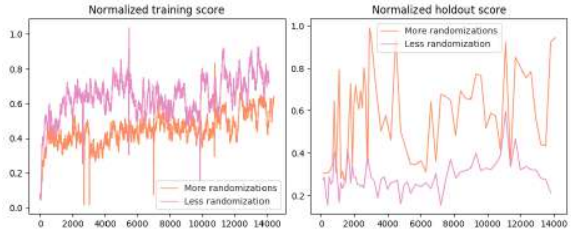


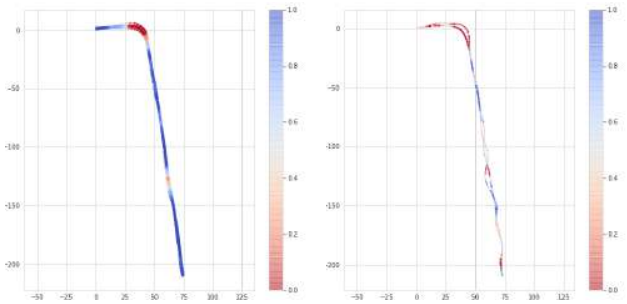
Fig. 5: Left: Episode scores obtained during training. The variant with less randomization is easier and faster to train. Right: On a holdout town with holdout weather better results are achieved by a model trained with more randomization.

B. Experiments in the real world

Perhaps unsurprisingly, in our real-world experiments we observed a high level of noise. Moreover, simulation results are a poor predictor of real-world performance. More precisely, in the set of policies above some threshold (of "decent driving") the correlation of simulation and real-wold scores is poor. This is perhaps the most evident instantiation of the sim-to-real gap (see also Section IV-D for a positive example of a offline metric correlating with real-world performance). These factors make it hard to put forward definitive conclusions. However, we were able to observe some trends and formulate recommendations for future research. Generally, we observe the positive influence of regularization and augmentation. It also seems beneficial to have an intermediate representation layer (semantic segmentation) and branching architectures (see Section IV-B-d).

Figure 3 summarizes the performance of our models in terms of the percentage of distance *driven autonomously* in all scenarios. See the project website `http://bit.ly/34xh7z4` for a more fine-grained presentation. Below we present a detailed per-experiment analysis.

a) Base experiments: The model `CONTINUOUS-PLAIN` exhibited very good performance and serves as a strong baseline for comparisons with other



CONTINUOUS-PLAIN DISCRETE-PLAIN

Fig. 6: Qualitative comparison of autonomy between a good model and a lagging one. The graphs show aggregation of few trials (test drives) by these models on the route `factory_city-tunnel-bt10`. The color depicts scale between full autonomy in all trials (blue) and human takeover in all trials (red). Graphs for other models and other routes are available on the project webpage `http://bit.ly/34xh7z4`.

variants discussed below. We aimed at creating a relatively simple model and training procedure. In other experiments we show that further simplifications deteriorate performance. Analogously to the experiment in simulation described above, we have verified the impact of training with fewer randomizations on real-world performance. As expected, the resulting model `CONTINUOUS-LOW-RAND` performs significantly worse, being in fact the worst model tested.

b) Discrete action space: This experiment aimed at measuring the impact of using a discrete distribution for the action space. The steering angles were discretized into unevenly distributed atoms. More of them were placed around 0 to improve smoothness of driving without increasing the action space (viz. [0., $\pm$0.01, $\pm$0.02, $\pm$0.03, $\pm$0.05, $\pm$0.08, $\pm$0.12, $\pm$0.15, $\pm$0.2, $\pm$0.25, $\pm$0.3, $\pm$0.4]; values are in radians). During training the action was sampled, while during evaluation a deterministic policy output the expected action (i.e. the sum of the atom values multiplied by their probabilities).

The resulting model `DISCRETE-PLAIN` performed badly in real-world evaluations, mostly due to severe side-to-side wobbling. We performed more experiments with discrete actions, with results being mostly weak (see `http://bit.ly/34xh7z4`).

c) Regularization: Improved performance in RL generalization when using regularization was reported in [6]. We evaluated using regularization in two experiments. In the first experiment, `DISCRETE-REG`, we fine-tuned `DISCRETE-PLAIN` by further training the model in a slightly altered setup: including l_2 regularization and reducing the policy entropy coefficient from 0.01 to 0.001. The resulting model behaved significantly better (for example the wobbling observed before almost disappeared). In the second experiment, the performance of the continuous model trained with regularization – `CONTINUOUS-REG` – was only slightly improved over `CONTINUOUS-PLAIN`.

d) Control via waypoints: Following the approach presented in [25], in experiment `WAYPOINTS-DISCRETE` we trained a model to predict the next waypoint instead of steering. Given a waypoint, low-level steering of the driving wheel is executed in order to reach this point. In simulation, this is realized by a PID controller, while in the case of the real car, we use a proprietary control system. To ensure similar performance in simulation and reality, we limit the action space of the RL agent to waypoints reachable by both of the controllers (this consists of points within a radius of 5 meters from the car). The action space is discrete – potential waypoints are located every 5 degrees between -30 and 30, where 0 is the current orientation of the vehicle.

In contrast to the experiments with direct steering, the continuous version of this experiment – `WAYPOINTS-CONTINUOUS` – was weaker and exhibited strong wobbling, even in simulation.

In this experiment we used a branched neural network architecture, again inspired by [25]. Namely, we use separate heads for each of the four high-level navigation commands (see Section III–e). Such architectures are considered to learn semantically different behaviors for the commands (e.g. *go straight* vs *turn left*) more easily and better handle command frequency imbalance. Our experimental results are in line with this interpretation – our models performed turns better than ones with the standard architecture. We expect that they will offer better general performance, which we plan to investigate in further research.

e) Dynamics randomizations: In [28], [27] dynamics randomization is pointed out as an important ingredient for successful sim-to-real transfer. In order to verify this in our context we introduced randomization to the following aspects of the environment: target speed, steering response (including a random multiplicative factor and bias), latency (the delay between observation and applying the policy's response to it), and noise in car metric observation (speed, acceleration, and wheel angle). Dynamics randomization parameters were sampled once at the beginning of each training episode.

For both experiments with dynamics randomization – `DYNAMICS-RAND-RNN` and `DYNAMICS-RAND-FFW` – the performance during evaluation on the real car was sub-standard. Somewhat surprisingly, the feed-forward model `DYNAMICS-RAND-FFW` performed slightly better than `DYNAMICS-RAND-RNN` using a GRU memory cell [5]. This is in contrast to literature, e.g. [28] which highlights importance of using memory cells: intuitively, an agent with memory should infer the dynamics parameters at the beginning of the episode and utilize them for better driving. We intend to further evaluate the possibility of using dynamics randomization for sim-to-real autonomous driving in future work. As a first step we will look for an explanation of the described mediocre performance. We speculate that this might be due to poor alignment of our randomizations with real-world requirements or overfitting when using high-capacity models with memory.

f) Auxiliary depth prediction: Auxiliary tasks are an established method of improving RL training (see e.g. [17]). Following that, in experiment `AUXILIARY-DEPTH` the model also predict the depth. The depth prediction is learned in a supervised way, along with RL training. This auxiliary

task slightly speeds up the training in simulation. However, in real-world evaluations it does not improve over the baseline experiments.

g) Segmentation only: Similarly to [25] we test the hypothesis that semantic segmentation is a useful common representation space for sim-to-real transfer. In experiment `SEMSEG-ONLY` the model takes in only segmentation as input and performs only slightly weaker than the baseline.

C. Selected failure cases

Besides major design decisions (as described in previous sections) there are a number of small tweaks and potential pitfalls. Below we show two examples, which we find illustrative and hopefully useful for other researches and practitioners.

1) Single-line versus double-line road markings: In initial experiments we have used CARLA's TOWN1 and TOWN2 maps, which feature only double-line road markings. When evaluated on real-world footage, policies trained only on the above were not sensitive to single-line road markings. This problem was fixed after introducing our custom maps, which feature single-line road markings.

2) Bug in reward function resulting in driving over the curb: Our reward function includes a term which incentivizes the agent to follow scenarios routes defined by checkpoints connected with straight lines. In one of our scenarios, based on the real-world testing area, the checkpoints were too sparse, resulting in one of the connecting lines going over a curb on a bend of the road. We noticed this only after doing a real-world test, where the car also exhibited similar behavior. This illustrates the well-known tendency of RL methods to overfit to idiosyncrasies of the reward function's design. Somewhat ironically, in this case our system overcame the sim-to-real gap and transferred the unexpected behaviour precisely. This suggest that methods with stronger generalization are required – we would hope they would generalize from the other bends the car could drive on without touching the curb.

D. Offline models evaluation

A fundamental issue in sim-to-real experiments is that good performance in simulation does not necessarily transfer to the real-world. This is aggravated by the fact that real-world testing is costly both in time and other resources. Inspired by [7] we introduced a proxy metric, which can be calculated offline and correlates with real-world evaluations. Namely, for the seven scenarios with prefix `factory_city` we obtained a human reference drive. Frame by frame, we compared the reference steering with the one given by our models by calculating the mean absolute error. We observe a clear trend (see Figure 7). While this result is still statistically rather weak, we consider it to be a promising future research direction. We present an additional offline evaluation metric (F_1) on the project website `http://bit.ly/34xh7z4`.

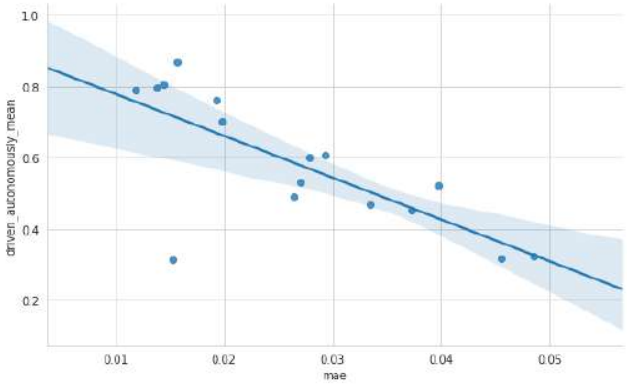


Fig. 7: Dependence of the mean of the *driven autonomously* metric on the mean absolute error with reference drives. Models utilizing waypoints are not included due to a different action space.

V. Conclusions and future work

We presented an overview of a series of experiments intended to train an end-to-end driving policy using the CARLA simulator. Our policies were deployed and tested on a full-size car exhibiting a substantial level of autonomy in a number of restricted driving scenarios.

The current results let us to speculate about the following promising directions: using more regularization, control via waypoints, and using offline proxy metrics. While we obtained poor results with memory-augmented architectures, we plan to investigate the topic further.

We also consider other training algorithms which use a replay buffer such as V-trace [14] and SAC [15]. The asymmetric actor-critic architecture presented in [29] and a generator-discriminator pair similar to the one in [3] can be also beneficial for training of driving polices. Another interesting and challenging direction is integration of an intermediate representation layer — for example a 2D-map or a bird's-eye view, as proposed in [4], [31], [12], [1]. Focusing RL training on fragments of scenarios with the highest uncertainty, see, e.g., [22] might improve driving stability. Integration of model-based methods similar to [24], [20] would be a desirable step towards better sample efficiency.

VI. Acknowledgements

We would like to thank Simon Barthel, Frederik Kanning, and Roman Vaclavik for their help and dedication during deployment on the physical car. We would also like to thank Tomasz Grel and Przemysław Podczasi for their contribution to the initial stage of the project. The work of Piotr Miłoś was supported by the Polish National Science Center grant UMO2017/26/E/ST6/00622. The work of Henryk Michalewski was supported by the Polish National Science Center grant UMO-2018/29/B/ST6/02959. This research was supported by the PL-Grid Infrastructure. We extensively used the Prometheus supercomputer, located in the Academic Computer Center Cyfronet in the AGH University of Science and Technology in Kraków, Poland. We managed our experiments using `https://neptune.ai`. We would like to thank the Neptune team for providing us access to the team version and technical support.

References

[1] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019.*, 2019.

[2] Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real world labels. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 4818–4824, 2019.

[3] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 4243–4250, 2018.

[4] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Model-free deep reinforcement learning for urban autonomous driving. *CoRR*, abs/1904.09503, 2019.

[5] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734, 2014.

[6] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289, 2019.

[7] Felipe Codevilla, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. On offline evaluation of vision-based driving models. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XV*, pages 246–262, 2018.

[8] Felipe Codevilla, Matthias Müller, Alexey Dosovitskiy, Antonio López, and Vladlen Koltun. End-to-end driving via conditional imitation learning. *CoRR*, abs/1710.02410, 2017.

[9] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

[10] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[11] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI Baselines. `https://github.com/openai/baselines`, 2017.

[12] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, and Jeff Schneider. Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *CoRR*, abs/1808.05819, 2018.

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv oloreprint arXiv:1711.03938*, 2017.

[14] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1406–1415, 2018.

[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1856–1865, 2018.

[16] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017.

[17] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[18] Stephen James, Andrew J. Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *CoRR*, abs/1707.02267, 2017.

[19] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12627–12637, 2019.

[20] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019.

[21] Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 6008–6014, 2019.

[22] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. In *British Machine Vision Conference 2017, BMVC 2017, London, UK, September 4-7, 2017*, 2017.

[23] Łukasz Kidziński, Sharada Prasanna Mohanty, Carmichael F. Ong, Zhewei Huang, Shuchang Zhou, Anton Pechenko, Adam Stelmaszczyk, Piotr Jarosik, Mikhail Pavlov, Sergey Kolesnikov, Sergey Plis, Zhibo Chen, Zhizheng Zhang, Jiale Chen, Jun Shi, Zhuobin Zheng, Chun Yuan, Zhihui Lin, Henryk Michalewski, Piotr Milos, Blazej Osinski, Andrew Melnik, Malte Schilling, Helge Ritter, Sean F. Carroll, Jennifer Hicks, Sergey Levine, Marcel Salathé, and Scott Delp. Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. In Sergio Escalera and Markus Weimer, editors, *The NIPS '17 Competition: Building Intelligent Systems*, pages 121–153, Cham, 2018. Springer International Publishing.

[24] Kendall Lowrey, Aravind Rajeswaran, Sham M. Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[25] Matthias Mueller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. Driving policy transfer via modularity and abstraction. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, pages 1–15, 2018.

[26] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *International Conference on Computer Vision (ICCV)*, 2017.

[27] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[28] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 1–8, 2018.

[29] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. In *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018.

[30] Dean Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, pages 305–313, 1988.

[31] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep imitative models for flexible inference, planning, and control. *CoRR*, abs/1810.06544, 2018.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015*

- *18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, pages 234–241, 2015.

[33] Andrei A. Rusu, Matej Vecerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, pages 262–270, 2017.

[34] Fereshteh Sadeghi. Divis: Domain invariant visual servoing for collision-free goal reaching. In *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019.*, 2019.

[35] Fereshteh Sadeghi and Sergey Levine. CAD2RL: real single-image flight without a single real image. In *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017.

[36] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. Sim2real viewpoint invariant visual servoing by recurrent control. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4691–4699, 2018.

[37] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, Jan 2017.

[38] Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, pages 237–252, 2018.

[39] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[40] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics, Results of the 11th International Conference, FSR 2017, Zurich, Switzerland, 12-15 September 2017*, pages 621–635, 2017.

[41] Niko Sünderhauf, Oliver Brock, Walter J. Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, and Peter Corke. The limits and potentials of deep learning for robotics. *I. J. Robotics Res.*, 37(4-5):405–420, 2018.

[42] Bowen Tan, Nayun Xu, and Bingyu Kong. Autonomous driving in reality with reinforcement learning and image translation. *CoRR*, abs/1801.05299, 2018.

[43] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In Tom Howard Hadas Kress-Gazit, Siddhartha Srinivasa and Nikolay Atanasov, editors, *Robotics: Science and System XIV*, 2018.

[44] Josh Tobin, Lukas Biewald, Rocky Duan, Marcin Andrychowicz, Ankur Handa, Vikash Kumar, Bob McGrew, Alex Ray, Jonas Schneider, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Domain randomization and generative models for robotic grasping. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 3482–3489, 2018.

[45] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033, 2012.

2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

What data do we need for training an AV motion planner?

Long Chen*, Lukas Platinsky*, Stefanie Speichert*, Błażej Osiński,
Oliver Scheel, Yawei Ye, Hugo Grimmett, Luca del Pero, and Peter Ondruska

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561723

***Abstract*— We investigate what grade of sensor data is required for training an imitation-learning-based AV planner on human expert demonstration. Machine-learned planners [1] are very hungry for training data, which is usually collected using vehicles equipped with the same sensors used for autonomous operation [1]. This is costly and non-scalable. If cheaper sensors could be used for collection instead, data availability would go up, which is crucial in a field where data volume requirements are large and availability is small. We present experiments using up to 1000 hours worth of expert demonstration and find that training with 10x lower-quality data outperforms 1x AV-grade data in terms of planner performance (see Fig. 1). The important implication of this is that cheaper sensors can indeed be used. This serves to improve data access and democratize the field of imitation-based motion planning. Alongside this, we perform a sensitivity analysis of planner performance as a function of perception range, field-of-view, accuracy, and data volume, and reason about why lower-quality data still provide good planning results.**

I. INTRODUCTION

While human drivers can skillfully navigate complex and varied scenarios involving multiple traffic agents safely, motion planning remains one of the hardest problems in autonomous driving. This has motivated the recent interest in planning approaches for Autonomous Vehicles (AVs) based on imitation-learning [1], [2]. These methods learn to mimic human behaviour from real-world human driving examples.

These approaches are extremely data-hungry, due to the complexity of the traffic scenarios and their variety: the so-called *long tail* of rare events. Some methods attempt to reduce the need for large volumes of human examples by employing data augmentation techniques, or by synthesizing rare cases in simulation [3]. However, just as the advancements in object detection and classification required large, real-world data sets (e.g., [4], [5]), we propose that motion planning also requires *a large corpus of real human driving data*. The problem is that these data are not readily available, and capturing it typically requires a fleet of vehicles equipped with expensive AV-grade sensors, the same that the AV needs to operate in autonomous mode (so that the planner can use the same perception system both at train and test time [1]). This significant barrier to entry stifles progress in the field.

We ask ourselves: do we really need a fleet with AV-grade sensors to collect training data for an AV motion planner? If vehicles equipped with commodity sensors were sufficient, data would be more readily available and this research problem could be democratized.

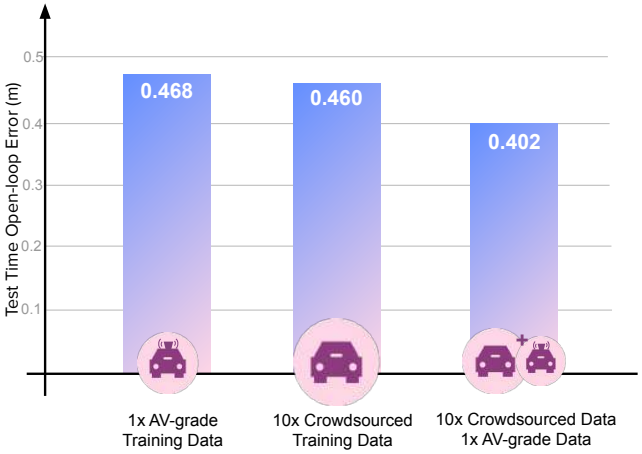


Fig. 1: Large data sets of expert driving demonstrations for training motion planners need not come from expensive fleets of AVs. We evaluate motion planner error rate (lower is better), and observe that a planner trained on expert demonstrations collected using AVs (AV-grade, left) has inferior performance to one trained on larger volumes of lower-quality data (middle). Combining the two data sources further improves performance (right), as this bridges the domain gap between the sensors used for training and those used by the AV at test time. This opens the door to crowd-sourced data collection using more affordable sensor configurations than those on AVs.

In this paper, we investigate the data properties we need to train an AV planner for urban operation, by comparing the performance of a state-of-the-art machine-learned planner as we train it on expert demonstration data with varying levels of quality. Modern planning approaches ([1], [6]) take as input the output of a vehicle's perception system (*perception output*) containing the 3D positions of other traffic agents. The supervision is provided by the human driving the vehicle. By taking AV-grade perception output and progressively limiting its key functional properties, like range and field-of-view (FoV), we simulate the lower-quality data that we could get from a variety of commodity sensors. Our experiments with this data inform what quality we need (and hence which sensors we need), and help us understand the trade-offs between data quantity and quality.

Our key contributions are as follows:

- We show that training on 10x more data with lower-

*Equal contribution
Authors are with Lyft Level 5 self-driving division.
Stefanie Speichert is also affiliated with the University of Edinburgh.
Project website: https://planning.l5kit.org

quality improves on 1x AV-grade data (see Fig.1). This has an important implication: we can collect expert demonstration data for training using vehicles equipped with sensors that are much cheaper than the AV-grade. This opens the door to scalable, cost-effective ways to collect large volumes of human driving examples for teaching AV systems how to drive.
- We present a sensitivity analysis showing which aspects of data quality are most important for planner performance and why quantity is more important than quality. We derive this from an extensive quantitative evaluation supported by attention-based analysis.

II. RELATED WORK

In this paper we build on recent advances in modular machine-learned planning systems.

End-to-end vs modular approaches. Current motion planning methods can be broadly categorized into end-to-end and modular (also known as mid-to-mid). End-to-end methods consume raw sensor data and output steering commands. A notable example is imitation learning for lane following [2], and learning end-to-end driving from simulation [7]. We refer to [8] for a broad review. Modular methods (e.g. [1]) subdivide the problem into sub-tasks (typically, perception, prediction, planning and control), each feeding on the output of the previous one. Sub-tasks are naturally more contained and easier to solve, and having intermediate outputs makes it easier to interpret the final output. More recently, Zeng et al. [9] proposed to bridge the two approaches with an end-to-end architecture producing (and trained on) intermediate outputs. Here, we use modular approach, where the planning module can transfer across different sensor configurations as it takes as input a shared representation, i.e. the output of the perception system.

ML planners. An overview of classical planning approaches, including for example expert systems, can be found in [10]. Recently, there has been interest in machine-learning (ML) approaches trained on expert demonstration [1], [6], which have the potential to scale with the data.

Relationship between planning and perception. Other work has studied the impact of perception quality on planning, either using involved hand-crafted features (e.g. the nuScenes Detection Score in [11]), or as a function of the performance of the planner [12]. While the latter work is relevant, our goal is not to propose a metric to evaluate perception performance, but rather to find what sensors we need to collect training data for planning. Wong et. al [3] simulate perception output for testing the AV planning system. Their focus is on synthesising realistic perception output as if it were captured by AV sensors, and not different levels of perception quality for simulating capturing data from cheaper sensors like we do here.

Training with different levels of supervisions. In our work, we train on a huge amount of lower-quality data and fine-tune on a small amount of high-quality data. This relates to self-supervised learning, which has recently revolutionized natural language processing [13], [14] and is in the process of radically changing computer vision [15], [16]. These approaches employ a pre-training phase, in which a neural network is trained on a large collection of unlabeled text/images, and then fine-tuned for the target task on a magnitude smaller collection. Our work is also related to weakly supervised learning, where training happens on data with noisy or incomplete supervision, for example training object detectors on entire images rather than manually annotate 2D bounding boxes [17]). In our case, what changes is the quality of the sensors used to collect the training data.

There are several transfer learning techniques to tackle domain shift between training and testing (e.g. [18]). One option is fine-tuning, which is widely used in literature, e.g. [19], [20]. Another is domain adaptation, i.e. aiming to explicitly reduce the domain gap between domains, often used in computer vision domains, for example for synthesizing examples in new domains or style transfer [21], [22], [23]. However, little work has been done to study transfer learning for motion planning.

Data availability. Much previous work relied on proprietary data [1]. Available datasets for planning are few and of moderate size, e.g. [24], [11], pointing to the data availability problem mentioned in earlier sections. In our experiments, we use the recent Lyft dataset [25], containing 1000 hours of expert demonstrations collected by AVs in urban settings.

III. METHODOLOGY

To understand what type of data we can use to train an AV motion planner, we start by asking two questions: what type of data do we need (*quality*) and how much (*quantity*)?

By data quality we mean the accuracy and robustness of the perception system that the planner uses as input both at test and train time [1]. The perception sytem outputs the 3D positions of other traffic agents like cars and pedestrians (**traffic agents**), which provides crucial context, as these positions influence the trajectory followed by the human driver on which the planner is trained. Quality is driven by the sensors installed on the vehicle, for example, an expensive AV-grade LIDAR can estimate depth much more accurately than a commodity LIDAR, while radars have longer range than a commodity camera system.

To understand data quality requirements, we propose to compare the performance at test time of a state-of-the-art ML-planner as we train it on input data collected with different sensors. However, this would require building and deploying a variety of different sensor configurations, and collecting enough training data with each of them. Instead, we propose to take a dataset of expert driving examples with corresponding AV-grade perception output (Fig. 2), and then simulate what this data would look like if it were collected by a wide variety of cheaper sensors. We do this by altering three key dimensions of the perception data:

1) **Range**: The maximum distance that sensors can see objects, e.g. 40m, 30m, etc.
2) **Field-of-view (FoV)**: How wide sensors can see, e.g. 90°, 180°)

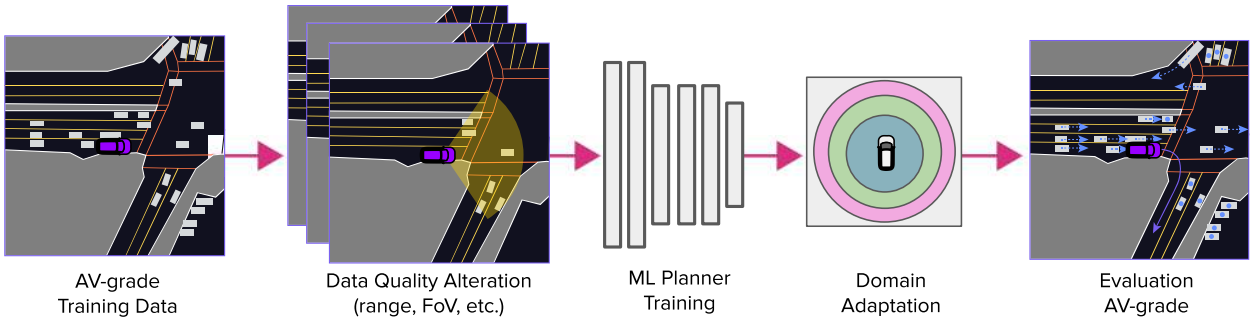


Fig. 2: An overview of our methodology (sec. III). Firstly, we collect expert demonstrations using AV-grade sensors, and extract perception information (which contains agent tracks). Secondly, we simulate what this data would look like if collected with lower-quality sensors by reducing the maximum range and field-of-view of the perception information. We then compare the performances of ML planners trained on these training sets with varying quality. This lets us do a comprehensive evaluation without having to build or deploy a wide variety of sensor configurations. To address domain shift, we fine-tune the planner on small amounts of data collected with AV-grade sensors, which is what the planner uses at test time.

3) **Geometric accuracy**: How accurate the perception of nearby vehicles is, e.g. the positional and rotational error of detected agents

We therefore start from a dataset of expert-driven examples with AV-grade quality perception, then alter the perception quality along one or more of these three dimensions, and finally train a planner using these data. We repeat this process with varying degrees of quality alteration (see Fig. 2). A nice property of using the same training samples for all experiments is that it allows us a fair comparison: the difference in performance is only due to difference in quality, and not, for example, due to one dataset containing more diverse examples than the other.

At test time we always use AV-grade input since, while training data can come from many sources, the goal is to deploy the planner on an actual AV. This introduces domain shift, since the input representation has a different distribution at test time (AV-grade) compared to training (lower-quality). Addressing this is an important part of our methodology. Having investigated data quality with this procedure, we conclude with an analysis on the relationship between data quantity and quality requirements (Sec. IV).

In what follows, we discuss the ML planner we use in our experiments (Sec. III-A), how we alter data quality (Sec. III-B), and how we address domain shift (Sec. III-C).

A. ML Planner

For our experiments we use a state-of-the art ML motion planner trained via imitation learning, similar to e.g. ChauffeurNet [1]. The input representation to the planner is a birds-eye-view rasterization of the current driving scene centered around the *ego vehicle* (the vehicle carrying the sensors). Agent tracks are rendered as 2D bounding boxes on top of the semantic map of the area, e.g. lane geometry, crosswalks, etc. (see Fig. 3). The network predicts the trajectory to follow for the next T time steps, which we denote by $p = (p_1, \ldots, p_T)$. The training loss minimises the L2 distance between p and the trajectory $\hat{p}$ followed by the human expert:

$$l = \sum_{t=1}^{T} |p - \hat{p}|_2 \tag{1}$$

We use a ResNet-50 backbone. The network outputs a trajectory for a 1.2 second ($T = 12$ steps) prediction horizon. During training we add synthetic perturbations by alternating the ground-truth trajectories to the left or right side using Ackerman steering [26] for realistic kinematics. This was shown to achieve better generalisation, which is needed for closed-loop testing [1].

B. Altering data quality

Range and field-of-view: We reduce the maximum range of the sensors by removing portions of the agent tracks that are beyond a given distance in the AV-grade data (e.g. 20m,

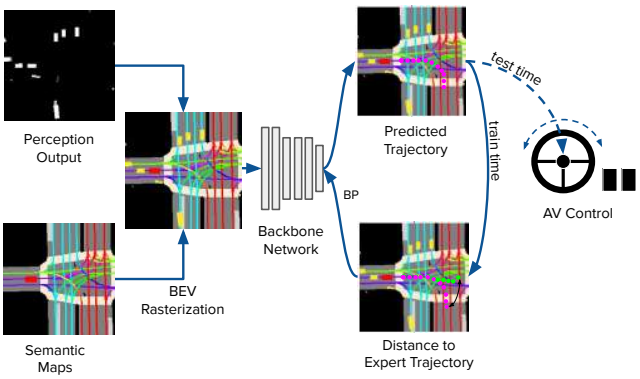


Fig. 3: We use a state-of-the-art ML planner (Sec. III-A) which takes as input a mid-level representation constructed by rasterizing the agent tracks detected from the perception system on top of an HD semantic map. The network predicts a trajectory for the AV to follow, minimising the distance from the trajectory followed by the expert at training time.

30m, etc.). We reduce the field-of-view (e.g. 90°, 180°) by removing agent tracks outside of it (compare Fig. 2). The field-of-view is centered along the forward direction of the ego vehicle, e.g. 90° degrees denoting 45° on each side.

Geometric accuracy: We add noise to the position and rotation of the AV-grade perception tracks using noise models approximating the accuracy we would expect from cheaper sensors.

Specifically, we add positional noise to entire agent tracks by applying random offsets to (x,y) agent coordinates in an agent-centered coordinate system. We measure the extent of the positional noise in terms of Intersection over Union (IoU), which is a standard metric for evaluating the accuracy of agent tracks (e.g. [27]). We specify an IoU noise level that we would like to alter the agent accuracy to. We then add random positional noise such that the noisy position will have the given IoU overlapping with the true position of the agent in the AV-grade data. More specifically, we apply random noise along the longitudinal axis using a uniform distribution, and add noise along the horizontal axis so that the specified IoU level is met. We sample this value once per agent track, and apply it to all agent positions in the track (this is equivalent to shifting the entire track). This is more realistic than, say, applying i.i.d noise to consecutive agent positions within the same track, as tracks from any sensor are typically processed with some form of smoothing.

For rotational noise, we add a random offset to the angle between the agent direction relative to ego, which we sample from $r \cdot \mathcal{N}(0,\, 1)$, where r is the maximum rotational noise we use in the experiments.

C. Addressing domain shift

Using an input representation with different properties at training and test time introduces domain shift. While our representation is the same in both cases (rasterised BEV, Fig. 2), the input agent tracks follow a different distribution and have different levels of noise.

Here, we fine-tune the planner on a relatively small amount of AV-grade data. Intuitively, we first learn how to plan from a large corpus of training data collected with low-quality sensors. Fine-tuning allows us to transfer this wealth of knowledge for use on the AV, which uses different, better sensors at test time. This is an integral part of our proposal of leveraging commodity sensors to collect human driving data. In our experiments we only fine-tune for 2 epochs and use 1/10th of the original base learning rate to avoid overfitting to the small AV-grade dataset.

IV. RESULTS

In this section we investigate the impact of training data quality for an ML planner (see Sec. IV-C), and the relationship between quantity and quality (see Sec. IV-E). We train our ML planner on either AV-grade data or lower-quality data, and always test on AV-grade data (unless specified otherwise), as our goal is ultimately to deploy the ML planner on a modern AV irrespective of the data it is trained on. We use standard metrics for this domain (Sec. IV-B).

A. Dataset

Our experiments are conducted on the Lyft Level 5 Prediction Dataset[1] [25] which contains > 1000h of expert driving demonstrations with corresponding AV-grade perception output. It was collected in the Palo Alto area by a fleet of 20 AVs, each having 7 cameras, 3 LiDARs, and 5 radars. The data is provided in independent chunks of 25s called *evaluation scenes*, and perception output is refreshed every 10 Hz. In all our experiments, we use the first 10h, 100h and 1000h data from the L5 training dataset for training, and the original validation dataset for testing, and let our planner predict at a rate of 10Hz, i.e. every time a new perception observation is available (we call these *steps*).

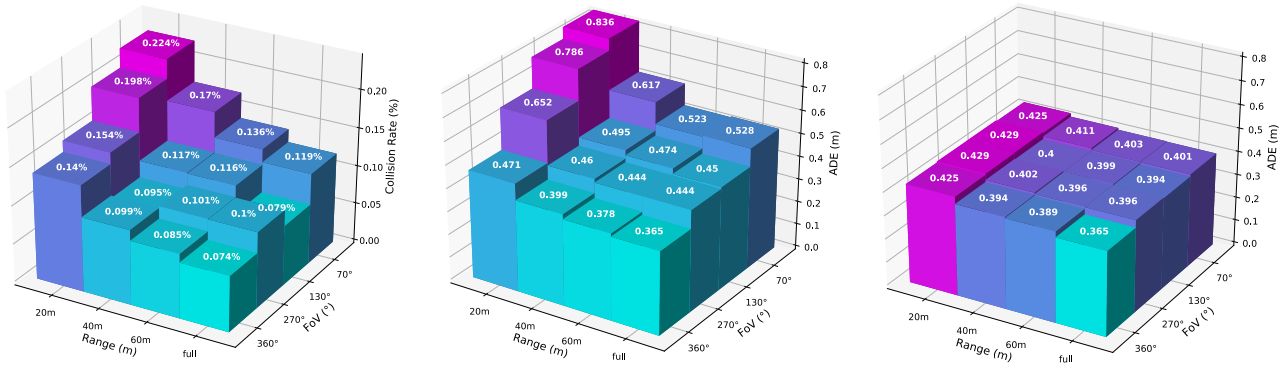
B. Metrics

1) Collision rate (closed-loop evaluation): In closed-loop evaluation we allow the ML planner to control the ego vehicle for an extended period of time. Specifically, at time $t = 0$ we use the planner to predict the ego location at timesteps $t_1, t_2 ... t_n$ (sec. III-A), and advance the ego to the location at t_1. We then feed the planner a new BEV generated at this location to obtain a new prediction. We continue doing this until we reach the end of the evaluation scene, unless the ego collides with an agent or the planner prediction deviates from the ground truth expert trajectory too much where the perception is not reliable anymore. In these last two cases, we stop evaluating the scene. The metric we report for closed-loop is the *collision rate*, i.e. the total number of collisions divided by the total number of steps in the dataset.

2) ADE (Open-loop error): Unlike in closed loop evaluation, here the planner makes a prediction for each step in a scene independently, and always starts from the ground-truth expert position (no unrolling). We measure the difference between the predicted trajectory and the expert trajectory over the 1.2s prediction horizon using (1), and average over all steps - this is called Average Distance Error (*ADE*). Such open-loop error is commonly used in the field ([1], [9]). While it does not provide a holistic view of whether the planner generalises well to the situation when it actually has control over the car like in closed-loop [28], it is less noisy and much more robust to problems like non-reactivity of the agents and deviations in position impacting perception.

3) Agent influence: Agent influence measures the impact of a specific agent track around the ego vehicle on the output of the planner. It is similar to [12] and helps us understand how this influence varies as a function of the relevant data quality dimensions (Sec. III-B). For example, the distance of influential agents from the ego vehicle impacts the range we need from the sensors, while their relative position to the ego vehicle impacts FoV. We compute influence for agent a as follows: at test time, we first use the planner to predict a trajectory p as usual. We then generate a second raster by masking agent a and let the planner predict a new trajectory p_a. We define agent influence to be $||p - p_a||_2$, which intuitively indicates how different the planner would

[1] https://self-driving.lyft.com/level5/data/



(a) Collision rate (b) ADE (c) ADE (fine-tuned)

Fig. 4: (a)-(b) Altering range and FoV of the training data (Sec. IV-C) impacts performance at test time, with similar trends on both collision rate (a) and ADE (b). There is a significant gap for data with 20m range, while for larger ranges the gap is much smaller. Performance changes smoothly as we alter FoV. (c) If we fine-tune (FT) on a small quantity of AV-grade data (10 hours), the gap between training entirely on AV-grade is now much smaller, showing that we can better leverage low quality data if we resolve domain shift (Sec. III-C). Compare this to (b), where we run the same experiment without fine-tuning.

behave if it could not see a. We further define an agent as *very influential* if this L2 distance is >0.1, and *slightly influential* if between 0.1 and 0.01. We compute this for all agents and all steps in the validation dataset.

C. Impact of data quality

To measure the impact of data quality, we alter the AV-grade perception tracks in the training data along the three dimensions mentioned in Sec. III. For range and FoV (Sec. III-B) we use the following buckets:

- Range: 20m, 40m, 60m, full range (AV-grade)
- FoV: 70°, 130°, 270°, 360° (AV-grade)

We train the planner on all possible permutations of range and FoV (16 in total) and report collision rate and ADE in Fig. 4. All results are using the same training set of 100 hours (we always train for 15 epochs), and are evaluated on the full Lyft validation set. For both metrics, performance increases significantly when the sensor range goes from 20m to 40m, and the FoV from 70° to 130°. However, longer range or wider FoV do not significantly impact performance, and sensors that can achieve 40m range and 130° FoV can already provide valuable training data.

Next, we analyse geometric accuracy (Sec. III-B) by using the following buckets:

- Positional error: 0.1 IOU, 1.0 IOU (zero error)
- Rotational error: 30°, 0°(zero error)

The results in Fig. 6c show that the planner is quite robust to this type of orientation and positional error (1 cm difference in ADE at test time). For this reason, we do not explore geometry accuracy any further in the next experiments.

Our agent influence metric allows us to gain further insights in these results. When training on AV-grade data, we note that the most influential agents are within the 40m range from the ego vehicle (Fig. 6a). This is consistent with Fig. 4, where training on more than 40m brings negligible improvement. For FoV (Fig. 6b), agents in front of the ego are the most influential, but we can see a non negligible mass of important agents also at the back. This is aligned with our results in Fig. 4, where can see a noticeable improvement in performance when training on 360° compared to 270°. A qualitative example is shown in Fig. 7.

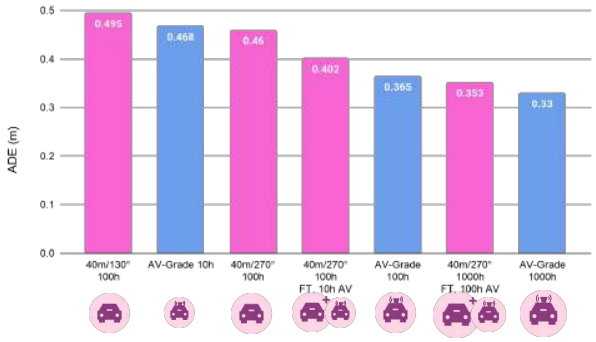
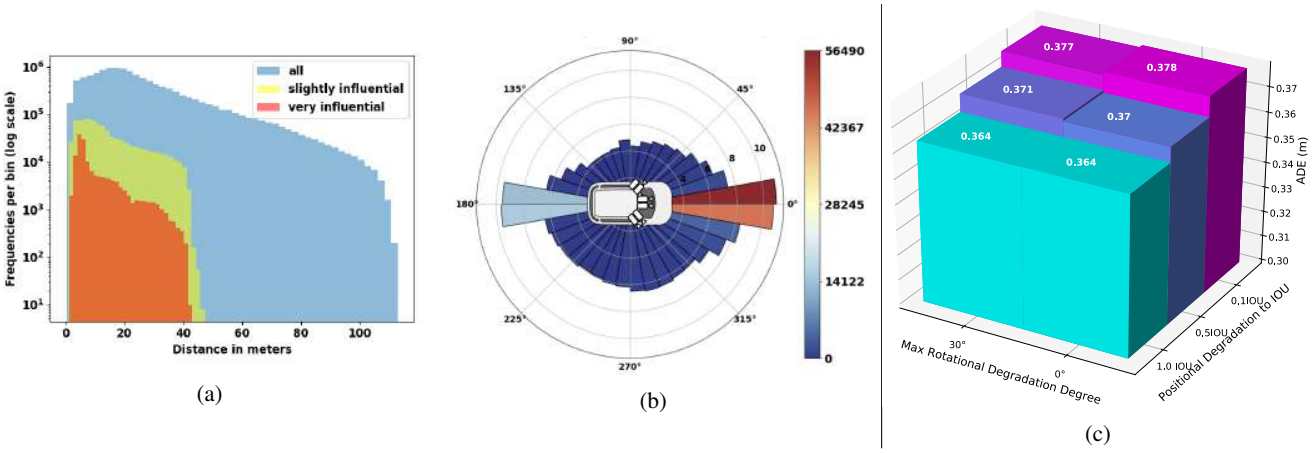


Fig. 5: Quantity vs quality. Training on 100 hours of low-quality data performs better than training on 10 hours of AV-grade for a wide variety of range/field-of-view combinations. If we fine-tune on 10 hours AV-grade, training on 100 hours of low-quality data approaches 100 hours of AV-grade (similarly, training on 1000h of low-quality and fine-tuning on 100h of AV-grade approaches 1000h of AV-grade).

D. Domain Shift

We repeat the experiment in Fig. 4b by fine-tuning each model on 10h of AV-grade data. Results in Fig. 4c show how much we can mitigate domain shift by exploiting a very limited amount of data from the target AV sensor configuration, and even models trained with only 20m range now become much more competitive.

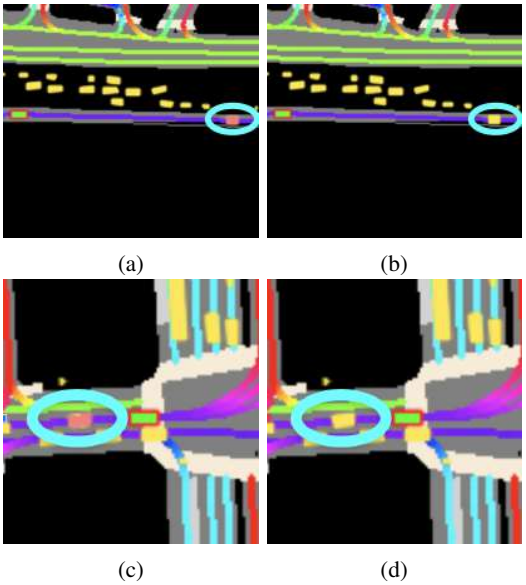


(a) (b) (c)

Fig. 6: (a)-(b) The proportion of influential agents (sec. IV-B) out of all agents observed in the validation set as a function of their distance to ego (a) and position in the vehicle's field-of-view (b). Most influential agents are within the 40m range and in front of ego (as expected). (c) ADE as we add positional and rotational error to the agent tracks in the training data (Sec. III-B). While test time performance degrades, the impact is negligible compared to altering range and FoV (Fig. 4).

E. Relationship between data quality and quantity

In this section we analyse the relationship between data quantity and quality. In Fig. 4 we saw a performance degradation when we alter quality while training on the same amount of data (100h). Fig. 5 clearly shows that training on 1000 hours low-quality data (10x) performs better at test time than 100 hours AV-grade (1x) for a variety of range/FoV configurations. As we fine-tune on additional 100h AV-grade data (Sec. III-C), we see that performance approaches training on 1000h AV-grade.



(a) (b) (c) (d)

Fig. 7: In scenario (a), a model trained on data with AV-grade range is mostly influenced by the only far agent (red) in front of the ego-vehicle (green), while a model trained on 20m range is not (b). We measure this using our agent influence metric (Sec. IV-B), i.e. how much the agent influences the planning decision. We observe the same phenomenon for FoV: when training on 360° (AV-grade), ego is influenced by agents behind (c), but is not when training on 270° (d).

V. Conclusion

We have shown that it is possible to train an AV planner on human expert demonstrations collected with sensors that are of lower quality than AV sensors. Our results suggest that data quantity trumps quality, and that it is better to have a lot of expert demonstrations with lower-quality, than a smaller amount of demonstrations that are AV-grade. For example, 100 hours with 40m range and 270° field-of-view data outperforms 10 hours of AV-grade quality (Fig. 5). Moreover, it approaches 100 hours AV-grade (the same amount) after fine-tuning on a small amount of AV-grade data, showing we can tackle domain shift. All in all this shows the promise of a crowd-sourcing approach for democratising the collection of training data for training AV motion planners.

We can use our results to inform the choice of sensors for collecting expert demonstration data, choosing the trade-off between sensor complexity (and cost) and expected performance in urban environments (Fig. 4c). The fact that several sensor configurations are competitive shows that we could combine data collected with a variety of heterogeneous sensors for training. Furthermore, we can apply the methodology presented in this paper to analyse data coming from different environments (e.g. highways) to understand the trade-offs between the sensor configurations in this setting.

In future work, we plan to experiment with data collected using real commodity sensors, rather than using our data-altering approach. Moreover, we will study how these results generalise to different areas and conditions.

ACKNOWLEDGMENTS

We would like to acknowledge and thank Sergey Zagoruyko for the valuable suggestions for this paper.

REFERENCES

[1] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[2] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *arXiv:1604.07316*, 2016.

[3] K. Wong, Q. Zhang, M. Liang, B. Yang, R. Liao, A. Sadat, and R. Urtasun, "Testing the safety of self-driving vehicles by simulating perception and prediction," *arXiv preprint arXiv:2008.06020*, 2020.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR*, 2009.

[5] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2014.

[6] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "Vectornet: Encoding HD maps and agent dynamics from vectorized representation," in *CVPR*. IEEE, 2020, pp. 11 522–11 530.

[7] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus, "Learning robust control policies for end-to-end autonomous driving from data-driven simulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1143–1150, 2020.

[8] A. Tampuu, M. Semikin, N. Muhammad, D. Fishman, and T. Matiisen, "A survey of end-to-end driving: Architectures and training methods," *arXiv:2003.06404*, 2020.

[9] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun, "End-to-end interpretable neural motion planner," in *CVPR*, 2019.

[10] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[11] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020, pp. 11 621–11 631.

[12] J. Philion, A. Kar, and S. Fidler, "Learning to evaluate perception models using planner-centric metrics," in *CVPR*, 2020, pp. 14 055–14 064.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*.

[15] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv preprint arXiv:2002.05709*, 2020.

[16] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020, pp. 9729–9738.

[17] Z. Ren, Z. Yu, X. Yang, M. Liu, Y. Lee, A. Schwing, and J. Kautz, "Instance-aware, context-focused, and memory-efficient weakly supervised object detection," *CVPR*, pp. 10 595–10 604, 2020.

[18] S. J. Pan and Q. Yang, "A survey on transfer learning," *Transactions on Knowledge and Data Engineering*, 2010.

[19] H. Noh, P. H. Seo, and B. Han, "Image question answering using convolutional neural network with dynamic parameter prediction," in *CVPR*, 2016.

[20] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh, "Vqa: Visual question answering," in *ICCV*, 2015.

[21] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *CVPR*, 2017.

[22] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," in *Int. Conf. on Machine Learning (ICML)*, 2018.

[23] X. Li, S. Liu, J. Kautz, and M.-H. Yang, "Learning linear transformations for fast arbitrary style transfer," *arXiv preprint arXiv:1808.04537*, 2018.

[24] M. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3d tracking and forecasting with rich maps," in *CVPR*, 2019, pp. 8740–8749.

[25] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska, "One thousand and one hours: Self-driving motion prediction dataset," *Conference on Robot Learning (CoRL)*, 2021.

[26] W. C. Mitchell, A. Staniforth, and I. Scott, "Analysis of ackermann steering geometry," in *SAE Technical Paper*. SAE International, 2006.

[27] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[28] F. Codevilla, A. M. López, V. Koltun, and A. Dosovitskiy, "On offline evaluation of vision-based driving models," in *ECCV*, 2018.

2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561666

SimNet: Learning Reactive Self-driving Simulations from Real-world Observations

Luca Bergamini*, Yawei Ye*, Oliver Scheel, Long Chen,
Chih Hu, Luca Del Pero, Błażej Osiński, Hugo Grimmett and Peter Ondruska+

***Abstract*— In this work we present a simple end-to-end trainable machine learning system capable of realistically simulating driving experiences. This can be used for verification of self-driving system performance without relying on expensive and time-consuming road testing. In particular, we frame the simulation problem as a Markov Process, leveraging deep neural networks to model both state distribution and transition function. These are trainable directly from the existing raw observations without the need of any handcrafting in the form of plant or kinematic models. All that is needed is a dataset of historical traffic episodes. Our formulation allows the system to construct never seen scenes that unfold realistically reacting to the self-driving car's behaviour. We train our system directly from 1,000 hours of driving logs and measure both realism, reactivity of the simulation as the two key properties of the simulation. At the same time we apply the method to evaluate performance of a recently proposed state-of-the-art ML planning system [1] trained from human driving logs. We discover this planning system is prone to previously unreported causal confusion issues that are difficult to test by non-reactive simulation. To the best of our knowledge, this is the first work that directly merges highly realistic data-driven simulations with a closed loop evaluation for self-driving vehicles. We make the data, code, and pre-trained models publicly available to further stimulate simulation development.**

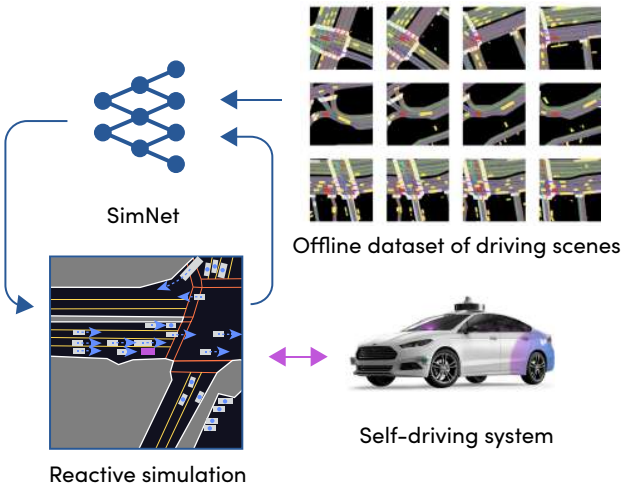


Fig. 1. The proposed trainable simulation system. We frame the simulation problem as reactive episode synthesis that can be used to validate the performance of a self-driving driving system.

I. INTRODUCTION

Self-Driving Vehicles (SDVs) have the potential to radically transform society in the form of safe and efficient transportation. Modern machine learning methods have enabled much of the recent advances in self-driving perception [2], [3], [4], [5], [6], prediction [7], [8], [9], [10] and planning [1], [11]. The availability of large datasets unlocked significantly higher performance compared to older, hand-engineered systems.

However, the problem of validating SDV performance remains still largely unsolved. Most industry players validate empirically by deploying their self driving systems to a fleet of vehicles accompanied by safety drivers. In the case of unusual or failure behaviours, the safety driver takes over. Observed issues serve as feedback to improve the system. However, this process is both expensive and time-consuming, requiring the collection of thousands or even millions of miles, depending on the system's maturity. It is also hard to replicate or directly compare the performance of different system versions, as it is impossible to experience exactly the same driving situation twice.

A common approach to mitigate some of these issues is *log replay*, where the movement of other traffic participants is replayed around the SDV in simulation as it happened when the log was collected. However, if the SDV's new actions differ from those when the log was collected, the traffic participants don't react to it, and thus the simulation becomes unrealistic and ineffective for validation. For example, even a slight braking during the log replay can result in an unrealistic collision with the trailing car due to non-reactivity. These unrealistic outcomes are a result of what is called *simulation drift*.

One way to implement simulation reactivity is by scripting traffic participant behaviour to follow certain rules. However, this is time consuming and still lacks the realism and fidelity of road testing, thus undermining the validation effort.

In this paper, we aim to create realistic simulated driving experiences just like those that an SDV would encounter in the real world. For example, when the SDV decides to slow down, the simulated vehicle behind it should react by either slowing or overtaking, just as it would in a road test. Additionally, agent behaviour should capture the stochastic multi-modality that we observe on the road.

To achieve this, we frame simulation as an ML problem, in which we generate driving episodes that need to be both *realistic* and *reactive* to SDV behaviour. We then present a

* Equal contribution.
+ Authors are with Lyft Level 5 self-driving division. Contact: pondruska@lyft.com.

Data, code and videos are available at `simulation.l5kit.org`

system leveraging high-capacity ML models trained on large amounts of historical driving data, Figure 1. We show the system's performance dramatically improves as the amount of data grows significantly narrowing the gap between road testing and offline simulation. At the same time, we show that it can help to identify issues in state-of-the-art planning systems. To the best of our knowledge, this is the first work that connects simulation and planning with large-scale datasets in a realistic self-driving setting.

Our contributions are four-fold:

1) The formulation of the self-driving simulation problem as an ML problem which seeks to generate driving episodes that are both realistic and reactive to SDV behaviour;
2) A simple machine-learned simulation system that can sample these episodes based on historical driving data trainable directly from traffic observations;
3) Qualitative and quantitative evaluation of the proposed method as a function of training data size, as well as, its usefulness in evaluating and discovering issues in a state-of-the-art ML planning systems.
4) The code and the pretrained models of the experiments to further stimulate development in the community.

II. RELATED WORKS

Our work is situated in the broader context of trajectory prediction, planning and simulation for autonomous vehicles.

Ability to predict future motion of traffic participants around the vehicle is important to be able to anticipate future necessary for planning. Classical methods include dynamic models [12], [13], [14], kinematic models [15], [16], [17], Kalman filter-based systems [18], [19], Monte Carlo sampling [20], [21], [22] and trajectory prototypes [23], [24], [25]. Today, deep learning methods for trajectory predictions are widely adopted. Notable examples include [7], [26], and [27]. Authors of [7] leverage bird's-eye view (BEV) rasters and a fixed set of future trajectory anchors. During training, the model learns displacement coefficients from those anchors along with uncertainties. TPNet [26] is a two stage network, where during the first stage the final future waypoint of the trajectory is predicted, starting from BEV semantic rasters. Then, proposals are generated to link past observations with this final waypoint and points near to it. We take inspiration from the above methods leveraging deep neural network architectures but intend to solve motion simulation instead of one-shot motion prediction. The key difference is that in motion simulation the sequence of traffic agent motion is generated one timestep at a time reflecting on both the intention of traffic agents themselves but also motion of other traffic participants and SDV.

Given the prediction of future motion of traffic participants SDV plans its own actions. Historically, this has been tackled by various methods optimising an expert cost function [28], [29], [30]. This cost function captures various desirable properties i.e. distance to other cars, comfort of the ride, obeying traffic rules etc. Engineering this cost function is, however, complex and time-consuming. Recently, novel methods [1], [11], [31] were proposed that frame the problem of planning as learning from demonstrations. Authors from [32] use an inverse reinforcement learning technique to learn from expert demonstrations. [33] deals with accumulating errors and presents a method trained entirely from data. A limitation of this model is that it directly predicts the visual output, which results in unnecessary errors, such as a car changing shape in consecutive frames.

In our work we do not try to build a planning system but to accurately evaluate performance of an existing one. In particular, we aim to explore performance of a recently proposed ML planning system [1]. This is a particularly appealing application given the novelty of ML planning systems in self-driving and their relatively unexplored performance. We find out that this particular method, while promising, is prone to causal confusion of imitation learning [34] that incorrectly associates motion of other cars with the desired action of the SDV. This issue is difficult to observe in non-reactive situation but becomes apparent when using a realistic reactive simulation.

Another way to perform simulation is to use an advanced driving simulator with agents controlled by hand-crafted rules. Notable examples of such driving simulators are SUMO [35] and CARLA [36]. A disadvantage of this solution is that hand-coded actors tend to be unrealistic and rarely present a wide enough variety of behaviours. Our aim is to achieve high realism by directly learning behaviour of other traffic participants from observed data. We show that these methods can be very powerful and their accuracy improves significantly with the amount of data used for training. As collecting these data is significantly easier than engineering a realistic simulation, this approach is much more scalable.

III. SELF-DRIVING SIMULATION AS A LEARNING PROBLEM

In this section, we formulate the simulation problem as a machine learning problem. Specifically, we aim to sample realistic driving experiences that the SDV would encounter in the real world, that also react to the SDV behaviour given by its control policy f. To help model this realism, we have access to historical driving scenes D that the capture observed behaviour of other traffic participants on top of a semantic map $\mathcal{M}$ in a variety of diverse driving scenarios.

Each driving episode of length T can be described as a sequence of observed states $s_1, s_2, ..., s_T$ with each state capturing the position, rotation, size and speed of all nearby traffic participants z:

$$s_t = \{z_t^1, z_t^2, ..., z_t^k\}. \tag{1}$$

This representation corresponds exactly to the output of the perception system, which turns raw sensor measurements into the vectorised detections of traffic participants and can then be fed to the SDV's control algorithm.

The SDV itself is modelled simply as one of these participants z^{SDV}, but unlike other traffic participants its dynamics are controlled by a known function f implementing the self-driving algorithm:

$$z_{t+1}^{\text{SDV}} = f(z_t^{\text{SDV}}, s_t). \tag{2}$$

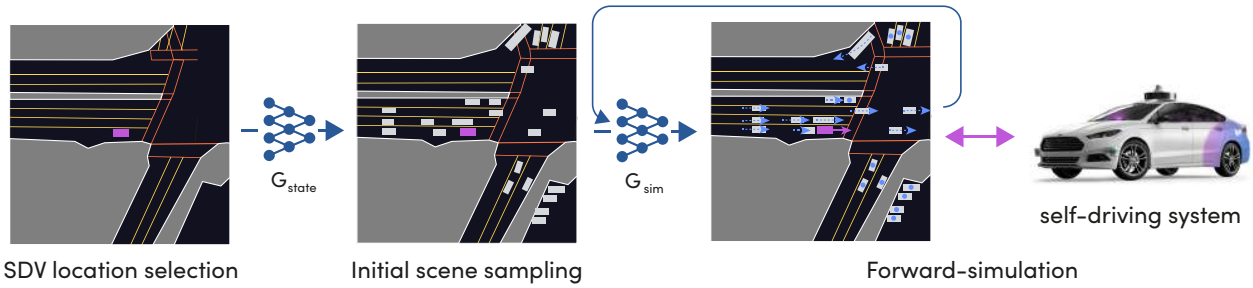


Fig. 2. Overview of the proposed simulation sampling process. To generate a new driving episode we first pick and sample an initial state capturing the positions of all traffic participants. Next, the state is forward-simulated with the traffic participants controlled by a neural network and the behaviour of SDV controlled by a self-driving control loop.

Generating new driving experiences can then be described as sampling from the joint distribution of the stochastic behaviour of other traffic participants and the deterministic SDV behaviour.

Note that this joint distribution is inherently non-deterministic. Given an initial state s_1 there are many possible ways that the future can unfold. At each moment in time, traffic participants must act and react to new information, such as attempts to merge, nudge, slow down, resulting in a complex set of driving behaviours. In the following section, we describe a method leveraging deep learning that can realistically sample such reactive episodes.

At the same time we aim to compute a certain set of metrics that describe the performance of the self-driving system, such as, the amount of *collisions*, *traffic rules violations* etc. An important property is the accuracy of these metrics or a ratio of false positive vs. false negative events. In an ideal simulation the amount of both is close to 0 but any deeper understanding about the correct attribution is valuable.

IV. GENERATING REALISTIC AND REACTIVE DRIVING EPISODES

In this section we describe an effective way to draw realistic and reactive driving episodes as defined in the previous section.

The sampling of driving episodes can then be formalised as a Markov Process with resulting probability distribution factorising as:

$$p(s_1, s_2, ..., s_T) = p(s_1) \prod_{t=2}^{T} p(s_t|s_{t-1}). \tag{3}$$

Furthermore, we assume the actions are locally independent for each participant conditioned on each previous state:

$$p(s_t|s_{t-1}) = \prod_{k=1}^{K} p(z_t^k|s_{t-1}). \tag{4}$$

This follows the intuition of real-world driving where participants act independently, each controlling their own behaviour, observing others and reacting to new information that becomes available after each time-step.

Both the initial state distribution $p(s_1)$ and participant policy $p(z_t^k|s_{t-1})$ are modelled by a separate neural networks G_{state} and G_{sim}. In particular, the participant transition policy is controlled by a neural network producing steering ϕ and velocity v

$$\phi^k, v^k \leftarrow G_{\text{sim}}(z_{t-1}^k, s_{t-1}) \tag{5}$$

that are used to update particular position of participant z^k.

Sampling from this process consists of executing three steps as summarised in Figure 2, and outlined in detail in the next subsections:

1) Initial SDV location l is chosen from all permissible locations on the map;
2) Initial state s_1 is drawn from the distribution of all feasible states. This state captures the total number and initial poses of all traffic participants;
3) A driving episode $s_2, ...s_T$ is generated via step-by-step forward simulation employing the participant's policy $p(z_t^k|s_{t-1})$ and self-driving control system f.

This formulation offers a high degree of flexibility, allowing one to tailor the properties of the resulting simulation:

- **Full simulation:** Executing all above steps results in generating new, never-experienced driving episodes from all locations.
- **Journey simulation:** By keeping the initial SDV location l fixed, we can synthesise many different initial conditions and driving episodes starting at that position.
- **Scenario simulation:** By using an existing historical state of interest as s_1, we can generate many resulting possible futures.
- **Behaviour simulation:** We can replace steering angle ϕ by hard-coding a specific path for them to follow. This forces a particular high-level behaviour of a traffic participant but still leaves a degree of reactiveness in execution. This is useful for simulating SDV behaviour in specific situations, e.g. being cut-off by another car.

A. Initial state sampling

To represent the state s around the self-driving vehicle, we leverage a bird's-eye view representation rendering positions of nearby traffic participants on top of a semantic map $\mathcal{M}$. This representation has proven to be an effective representation in recent motion prediction and planning works [37], [1]. One advantage is that it effectively captures both local

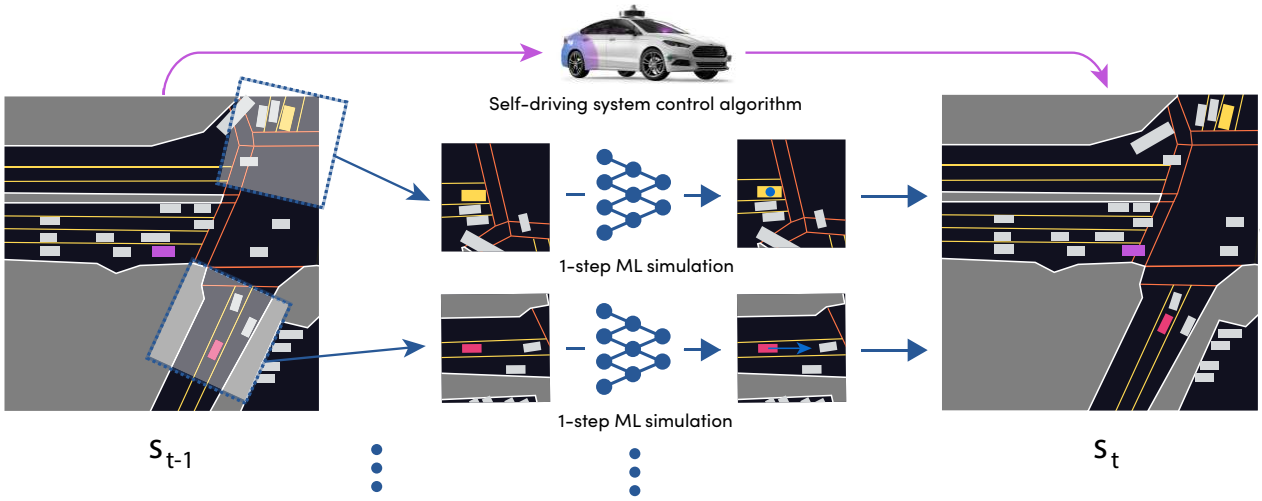


Fig. 3. Detail of the interactive state unroll. For all agents in the state s_{t-1} we independently run 1-step prediction to advance them. The self-driving car is controlled by control algorithm f. The new positions then form a new state s_t and the process repeats.

context and a variable amount of traffic participants in the form of a single image I_s.

To sample an initial state s_1 similar to our training distribution, we leverage conditional generative adversarial networks (cGANs) [38] conditioned on empty scenes capturing only the semantic map $I_{\mathcal{M}}$. This network is trained on pairs of $\{I_{\mathcal{M}}, I_s\}$ constituting the training dataset. Specifically, the generator network is trained to convert $I_{\mathcal{M}}$ into I_s and the discriminator to distinguish the synthetic states I_s from real ones. Upon convergence, the generator can create unlimited amounts of new synthesised states s_1 for any map location, indistinguishable from real ones, to seed the simulation.

To extract the final numerical positions and rotations of the vehicles $z_1, z_2, ...z_K \in s$, we use a connected components algorithm. For each connected component we compute the centroid and a minimum bounding box capturing its size.

B. Forward simulation

This step generates the full sequence $s_2, s_3, ..., s_T$. This generation happens one step at a time, executing policy $p(z_t^k|s_{t-1})$ for each traffic participant and SDV control policy f for the self-driving vehicle.

As the traffic participant policy we employ a model described in [37] consisting of a ResNet-50 backbone that takes bird's-eye-view rasterised states s_t around the traffic participants z_t^k as input, and a regression head predicting steering τ_t^k and velocity v_t^k. We train the model on past agent trajectories. In particular, we take all traffic participants from the training dataset D with observation history longer than 1s and train the model to predict their individual steering and velocity.

As illustrated in Figure 3, to compute a new state s_t the policy is invoked for every observed traffic participant z_k^t in the current state independently. Simultaneously, vehicle controls are triggered to obtain the SDV's new position:

$$z_t^{\text{SDV}} = f(s_{t-1}, z_{t-1}^{\text{SDV}}). \quad (6)$$

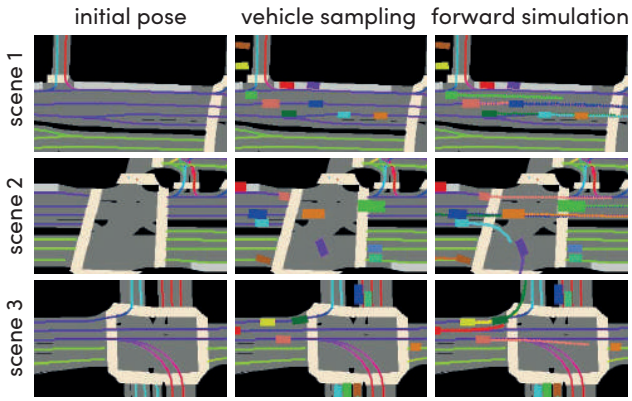


Fig. 4. An example of initial state and intended traffic participant trajectories. Each row shows a separate exemplar episode. From left to right: the initial scene sampled from SDV positions with agents being removed, the sampled traffic participants' positions, and the trajectory taken by each vehicle.

This then constitutes a new state s_t, and the process repeats until the entire episode is generated or simulation is interrupted, i.e. due to a simulated SDV collision.

V. EXPERIMENTS

Here we provide qualitative and quantitative evaluation of the proposed simulation system. In particular, we are interested in the system's ability to synthesise realistic initial states, forward-simulate full driving episodes, and to react to the SDV's behaviour. To evaluate it, we forward-simulate for 5 seconds across many scenes. Unrolling for 5 seconds is enough to capture interesting maneuver while still being able to collect ground truth annotations from the tracked agents in the dataset.

We capture the system's performance using two metrics. Both metrics are evaluated on 960 scenes, although these sets of scenes are disjoint.

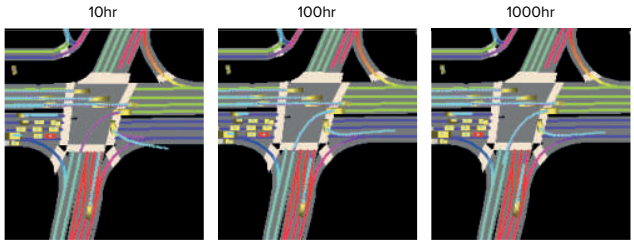


Fig. 5. Qualitative example of the simulation trained on various amount of data. More training data leads to more realistic simulations, with model predicted paths shown as blue lines and ground truth paths as purple lines.

	Displacement error [m]						
Method	0.5s	1s	2s	3s	4s	5s	**Reactivity**
Log replay	0	0	0	0	0	0	0
10 hr	0.58	0.95	1.76	2.57	3.40	4.28	0.95
100 hr	0.51	0.81	1.44	2.04	2.61	3.19	**0.97**
1000 hr	**0.49**	**0.76**	**1.32**	**1.87**	**2.42**	**2.99**	**0.97**

TABLE I

REALISM AND REACTIVENESS OF NON-REACTIVE AND REACTIVE SIMULATION TRAINED ON VARIOUS AMOUNTS OF DATA FOR VARIOUS SIMULATION HORIZONS. LOG-REPLAY IS PERFECTLY REALISTIC BUT NOT REACTIVE WHILE SIMNET IS BOTH REALISTIC AND REACTIVE.

Simulation realism: An average L2 distance between simulated agents and their ground-truth positions at different time steps into the future (displacement). For this experiment we initialise the state from a real-world log, and the SDV follows exactly the same path as it did in that log. A perfect simulation system should be able to replicate the behaviour of other agents as it happens in the log.

Simulation reactivity: We measure collision rate in synthetic scenarios where a static car is placed in front of a moving car, see the first two rows in Figure 6. This should not cause collisions in reality, and requires trailing cars to react by stopping. We report the number of scenes without a collision divided by the total number of scenes tested.

A. Implementation Details

We train and test our approach on the recently released Lyft Motion Prediction Dataset [39]. The dataset consists of more than 1,000 hours of dense traffic episodes captured from 20 self-driving vehicles. We follow the proposed train / validation / test split. We rasterise the high-definition semantic map included in the dataset to create bird's-eye view representations of the state, centered around each agent of interest to predict its future trajectories. This representation includes lanes, crosswalks and traffic light information. At crossings, lanes are not rendered if the controlling traffic light is red. As for agents, we focus our attention on vehicles (92.47% of the total annotated agents), pedestrians (5.91%) and cyclist (1.62%). We use rasters of size 224x224, a batch size of 64 and the Adam [40] optimizer in all our experiments. During evaluation, the whole pipeline takes around 400ms per frame with a modern GPU, which is acceptable for a non real-time constrained system.

B. Initial state sampling

A qualitative example of sampling the initial state and intended trajectory for each traffic participant is shown in Figure 4. We specifically focus on intersections as these situations give an opportunity to traffic participants to make a variety of decisions. As one can see, the generated states and scenes look realistic. Furthermore, learned trajectories effectively capture the variety of possible participant behaviours.

C. Forward simulation

The evaluation of SimNet demonstrates very good performance with respect to both simulation realism and reactiveness. The quantitative results can be found in Table I. The performance improves with the amount of data used for training. The qualitative comparison of models' realism is presented in Figure 5. Similarly, simulation reactiveness is presented in Figure 6.

D. Evaluating SDV with SimNet

The purpose of SimNet is to accurately evaluate the performance of the SDV. In this section we describe an experiment verifying that the model is indeed fit for the purpose.

We implemented a motion planner based on the state-of-the-art ChaufferNet [1]. The model's input and backbone are the same as for SimNet (see Section V-A). It predicts a future trajectory for the ego vehicle and is trained using behavioural cloning. Naive behavioural cloning suffers from the distribution shift between training and evaluation data. Similarly to [1], we alleviate this problem by introducing synthetic perturbations to the training trajectories.

We compared two ways to evaluate such a planner, based on log-replay and SimNet. The results are presented in Table II. The number of errors turned out to differ significantly in two categories: rear collisions and passiveness. This discrepancy raises the question of whether the reported errors reveal real mistakes of the planner or they are only artifacts of the incorrect methods of evaluation.

In order to determine this, we conducted a qualitative analysis of failure cases. In log-replay, the cases of rear collisions consisted of both false positives (when the ego was driving slightly slower than the reference trajectory and the chasing car did not accommodate for this), as well as

Planning metric	**Log-replay**	**SimNet**
Front collisions	2	2
Side collision	4	9
Rear collision	60	2
Displacement error	19	27
Passiveness	32	124
Distance to reference trajectory	2	2

TABLE II

PLANNING METRICS OF [1] WHEN EVALUATED USING NON-REACTIVE AND REACTIVE SIMULATION. NON-REACTIVE SIMULATION REPORTS VARIOUS ISSUES, SUCH AS, PASSIVENESS AS FALSE POSITIVE REAR COLLISIONS. THESE CAN BE PROPERLY IDENTIFIED USING REACTIVE SIMULATION.

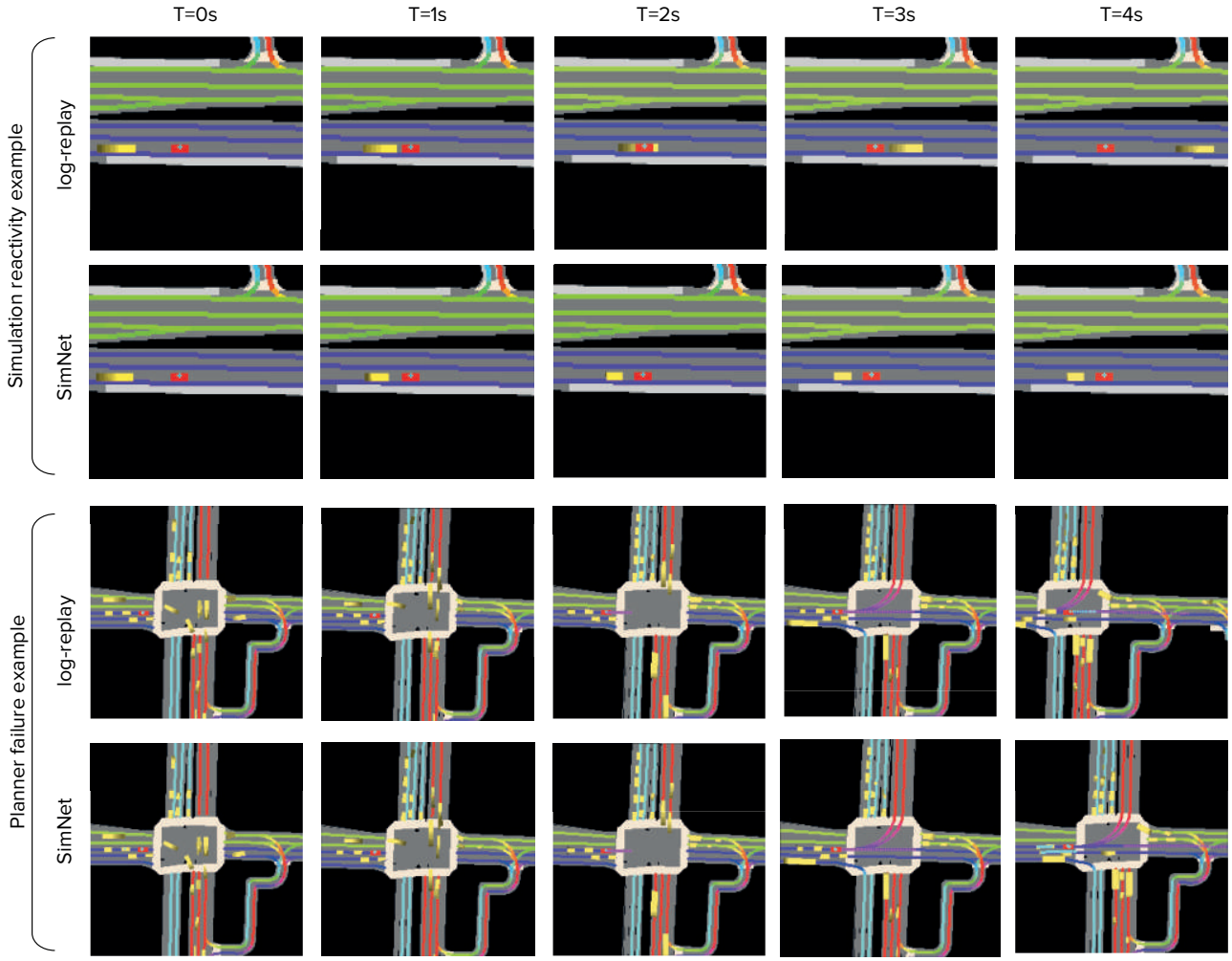


Fig. 6. The first two rows: a qualitative example of the reactivity evaluation - the agent controlled by SimNet [yellow] stopped behind the static vehicle [red], while the log-reply crashed into it without showing any reactivity. Last two rows: we found the reactivity of SimNet can expose causal confusion of ML planner - SDV waits for a slight movement of the chase car to start moving as would happen in log-replay. In reactive simulation this signal does not come and the SDV keeps waiting at the intersection. Best viewed in high-resolution.

true positives (when the ego was passive and not starting at an intersection at a green light). Both of these types of rear collisions disappear when evaluated in SimNet. This is to be expected, as in SimNet the chasing vehicle reacts to the static or slower ego. This could partially explain why SimNet reports higher passiveness errors compared to log-replay.

However, the increase in passiveness (from 32 to 124) is bigger than the total number of rear collisions (60). A qualitative investigation of the scenes uncovered an interesting failure mode of the ML planner: it would not start at an intersection if neighbouring agents are static (see example in Figure 6 below). This is a previously unreported instance of the causal confusion [41]. Moreover, it would not be possible to uncover it using log replay, because in such cases neighbouring agents (both in the front and behind the agent) will move as they did during log recording.

VI. CONCLUSIONS

We have presented an end-to-end trainable machine learning system that generates simulations of on-road experiences for self-driving vehicles. SimNet leverages large volumes of historical driving logs to synthesize new realistic and reactive driving episodes that can be used to validate SDV performance. The evaluation shows that SimNet achieves very good results in terms of both realism and reactivity. Moreover, using it for evaluating an ML planner has resulted in uncovering a previously unreported causal confusion of ChaufferNet [1]. Notably, we have confirmed that SimNet, which is also trained using imitation learning, exhibits the same failure mode. We consider addressing the issue of causal confusion to be an important further work aimed at improving both ML planners and simulators.

We believe this is an exciting step towards significantly decreasing the need for on-road testing in self-driving development, and the democratisation of the field. We hope the release of our system's source code will further stimulate development in ML simulation systems.

REFERENCES

[1] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[2] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[5] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[6] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[7] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," in *Conference on Robot Learning*, 2020.

[8] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "Desire: Distant future prediction in dynamic scenes with interacting agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[9] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine, "Precog: Prediction conditioned on goals in visual multi-agent settings," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[10] C. Tang and R. R. Salakhutdinov, "Multiple futures prediction," in *Advances in Neural Information Processing Systems*, 2019.

[11] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun, "End-to-end interpretable neural motion planner," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[12] M. Brännström, E. Coelingh, and J. Sjöberg, "Model-based threat assessment for avoiding arbitrary vehicle collisions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, 2010.

[13] C.-F. Lin, A. G. Ulsoy, and D. J. LeBlanc, "Vehicle dynamics and external disturbance estimation for vehicle path prediction," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 3, 2000.

[14] J. Huang and H.-S. Tan, "Vehicle future trajectory prediction with a dgps/ins-based positioning system," in *American Control Conference*, 2006.

[15] S. Ammoun and F. Nashashibi, "Real time trajectory prediction for collision risk estimation between vehicles," in *International Conference on Intelligent Computer Communication and Processing*, 2009.

[16] J. Hillenbrand, A. M. Spieker, and K. Kroschel, "A multilevel collision mitigation approach—its situation assessment, decision making, and performance tradeoffs," *IEEE Transactions on intelligent transportation systems*, vol. 7, no. 4, 2006.

[17] R. Miller and Q. Huang, "An adaptive peer-to-peer collision warning system," in *IEEE Vehicular Technology Conference*, 2002.

[18] H. Dyckmanns, R. Matthaei, M. Maurer, B. Lichte, J. Effertz, and D. Stüker, "Object tracking in urban intersections based on active use of a priori knowledge: Active interacting multi model filter," in *IEEE Intelligent Vehicles Symposium (IV)*, 2011.

[19] H. Veeraraghavan, N. Papanikolopoulos, and P. Schrater, "Deterministic sampling-based switching kalman filtering for vehicle tracking," in *IEEE Intelligent Transportation Systems Conference*, 2006.

[20] A. Broadhurst, S. Baker, and T. Kanade, "Monte carlo road safety reasoning," in *IEEE Proceedings. Intelligent Vehicles Symposium*, 2005.

[21] A. Eidehall and L. Petersson, "Statistical threat assessment for general road scenes using monte carlo sampling," *IEEE Transactions on intelligent transportation systems*, vol. 9, no. 1, 2008.

[22] M. Althoff and A. Mergel, "Comparison of markov chain abstraction and monte carlo simulation for the safety assessment of autonomous cars," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, 2011.

[23] C. Hermes, C. Wohler, K. Schenk, and F. Kummert, "Long-term vehicle motion prediction," in *IEEE intelligent vehicles symposium*, 2009.

[24] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank, "A system for learning statistical motion patterns," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 9, 2006.

[25] S. Atev, G. Miller, and N. P. Papanikolopoulos, "Clustering of vehicle trajectories," *IEEE transactions on intelligent transportation systems*, vol. 11, no. 3, 2010.

[26] L. Fang, Q. Jiang, J. Shi, and B. Zhou, "Tpnet: Trajectory proposal network for motion prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[27] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," *arXiv preprint arXiv:2007.13732*, 2020.

[28] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*. springer, 2009, vol. 56.

[29] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[30] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for bertha—a local, continuous method," in *2014 IEEE intelligent vehicles symposium proceedings*. IEEE, 2014, pp. 450–457.

[31] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.

[32] F. Behbahani, K. Shiarlis, X. Chen, V. Kurin, S. Kasewa, C. Stirbu, J. Gomes, S. Paul, F. A. Oliehoek, J. Messias *et al.*, "Learning from demonstration in the wild," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 775–781.

[33] M. Henaff, A. Canziani, and Y. LeCun, "Model-predictive policy learning with uncertainty regularization for driving in dense traffic," in *International Conference on Learning Representations*, 2018.

[34] P. de Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," *arXiv preprint arXiv:1905.11979*, 2019.

[35] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/

[36] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on Robot Learning*, 2017.

[37] H. Cui, V. Radosavljevic, F.-C. Chou, T.-H. Lin, T. Nguyen, T.-K. Huang, J. Schneider, and N. Djuric, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[38] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[39] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska, "One thousand and one hours: Self-driving motion prediction dataset," *arXiv preprint arXiv:2006.14480*, 2020.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] P. de Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

Urban Driver: Learning to Drive from Real-world Demonstrations Using Policy Gradients

Oliver Scheel, Luca Bergamini, Maciej Wołczyk, Błażej Osiński, Peter Ondruska
Woven Planet, Level 5
{firstname.lastname}@woven-planet.global

Abstract: In this work we are the first to present an offline policy gradient method for learning imitative policies for complex urban driving from a large corpus of real-world demonstrations. This is achieved by building a differentiable data-driven simulator on top of perception outputs and high-fidelity HD maps of the area. It allows us to synthesize new driving experiences from existing demonstrations using mid-level representations. Using this simulator we then train a policy network in closed-loop employing policy gradients. We train our proposed method on 100 hours of expert demonstrations on urban roads and show that it learns complex driving policies that generalize well and can perform a variety of driving maneuvers. We demonstrate this in simulation as well as deploy our model to self-driving vehicles in the real-world. Our method outperforms previously demonstrated state-of-the-art for urban driving scenarios – all this without the need for complex state perturbations or collecting additional on-policy data during training. We make code and data publicly available.

Keywords: Self-driving, Learning from Demonstrations, Planning, Simulation

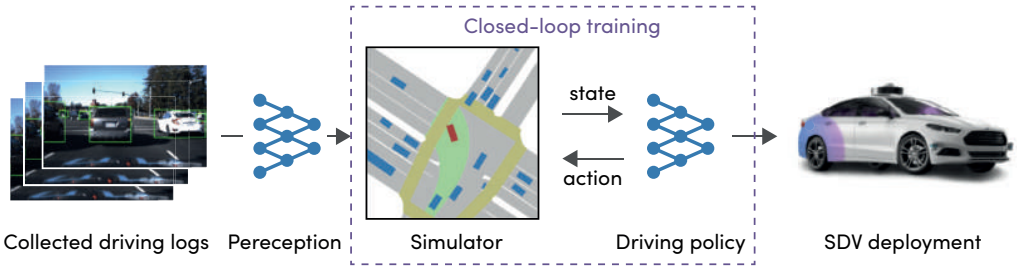


Figure 1: Overview of the proposed closed-loop training method for learning driving policies. We leverage large amounts of expert demonstrations and mid-to-mid representations to build a differentiable simulator supporting fast policy learning. With this simulator, we can effectively unroll model policies, and thus train the model closed-loop using a policy gradient method.

1 Introduction

Self-driving has the potential to revolutionize transportation and is a major field of AI applications. Even though already in 1990 there were prototypes capable of driving on highways [1], technology is still not widespread, especially in the context of urban driving. In the past decade, the availability of large datasets and high-capacity neural networks has enabled significant progress in perception [2, 3] and the vehicles' ability to understand their surrounding environment. Self-driving decision making, however, has seen very little benefit from machine learning or large datasets. State-of-the-art planning systems used in industry [4] still heavily rely on trajectory optimisation techniques with expert-defined cost functions. These cost functions capture desirable properties of the future vehicle path. However, engineering these cost functions scales poorly with the complexity of driving situations and the long tail of rare events.

Due to this, learning a driving policy directly from expert demonstrations is appealing, since performance scales to new domains by adding data rather than via additional human engineering effort.

5th Conference on Robot Learning (CoRL 2021), London, UK.

In this paper we focus specifically on learning rich driving policies for urban driving from large amounts of real-world collected data. Unlike highway driving [5], urban driving requires performing a variety of maneuvers and interactions with, e.g., traffic lights, other cars and pedestrians.

Recently, rich mid-level representations powered by large-scale datasets [6, 7], HD-maps and high-performance perception systems enabled capturing nuances of urban driving. This led to new methods achieving high performance for motion prediction [8, 9]. Furthermore, [10] demonstrated that leveraging these representations and behavioral cloning with state perturbations leads to learning robust driving policies. While promising, difficulty of this approach lies in engineering the perturbation noise mechanism required to avoid covariate shift between training and testing distribution.

Inspired by this approach, we present the first results on offline learning of imitating driving policies using mid-level representations, a closed-loop simulator and a policy gradient method. This formulation has several benefits: it can successfully learn high-complexity maneuvers without the need for perturbations, implicitly avoid the problem of covariate shift, and directly optimize imitation as well as auxiliary costs. The proposed simulator is constructed directly from the collected logs of real-world demonstrations and HD maps of the area, and can synthesize new realistic driving episodes from past experiences (see Figure 1 for an overview of our method). Furthermore, for training on large datasets reducing the computational complexity is paramount. We leverage vectorized representations and show how this allows for computing policy gradients quickly using backpropagation through time. We demonstrate how these choices lead to superior performance of our method over the existing state-of-the-art in imitation learning for real-world self-driving planning in urban areas.

Our contributions are four-fold:

- The first demonstration of policy gradient learning of imitative driving policies for complex urban driving from a large corpus of real-world demonstrations. We leverage a closed-loop simulator and rich, mid-level vectorized representations to learn policies capable of performing a variety of maneuvers.
- A new differentiable simulator that enables efficient closed-loop simulation of realistic driving experiences based on past demonstrations, and quickly compute policy gradients by backpropagation through time, allowing fast learning.
- A comprehensive qualitative and quantitative evaluation of the method and its performance compared to existing state-of-the-art. We show that our approach, trained purely in simulation can control a real-world self-driving vehicle, outperforms other methods, generalizes well and can effectively optimize both imitation and auxiliary costs.
- Source code and data are made available to the public[1].

2 Related Work

In this section we summarize different approaches for solving self-driving vehicle (SDV) decision-making in both academia and industry. In particular, we focus on both optimisation-based and ML-based systems. Furthermore, we discuss the role of representations and datasets as enablers in recent years, to tackle progressively more complex Autonomous Driving (AD) scenarios.

Trajectory-based optimization is still a dominant approach used in industry for both highway and urban-driving scenarios. It relies on manually defined costs and reward functions that describe good driving behavior. Such cost can then be optimized using a set of classical optimization techniques (A* [11], RRTs [12], POMDP with solver [13], or dynamic programming [14]). Appealing properties of these approaches are their interpretability and functional guarantees, which are important for AD safety. These methods, however, are very difficult to scale. They rely on human engineering rather than on data to specify functionality. This becomes especially apparent when tackling complex urban driving scenarios, which we address in this work.

Reinforcement learning (RL) [15] removes some complexity of human engineering by providing a reward (cost) signal and uses ML to learn an optimal policy to maximize it. Directly providing the reward through real-time disengagements [16], however, is impractical due to a low sample-efficiency of RL and the involved risks. Therefore, most approaches [17] rely on constructing a

[1]Code and video available at https://planning.l5kit.org.

simulator and explicitly encoding and optimising a reward signal [18]. A limiting factor of these approaches is that the simulator often is hand-engineered [19, 20], limiting its ability to capture long-tail real-world scenarios. Recent examples of sim-to-real policy transfer (e.g. [21], [22], [23]) were not focused on evaluating scenarios typical to urban driving, in particular interacting with other agents. In our work, we construct the simulator directly from real-world logs through mid-level representations. This allows training in a variety of real-world scenarios with other agents present, while employing efficient policy-gradient learning.

Imitation learning (IL) and Inverse Reinforcement Learning (IRL) [24, 25] are more scalable ML approaches that leverage expert demonstrations. Instead of learning from negative events, aim is to directly copy expert behavior or recover the underlying expert costs. Simple behavioral cloning was applied already back in 1989 [26] on rural roads, more recently by [27] on highways and [28] in urban driving. Naive behavioral cloning, however, suffers from covariate shift [24]. This issue has been successfully tackled for highway lane-following scenarios by reframing the problem as classification task [29] or employing a simple simulator [5], constructed from highway cameras. We take inspiration from these approaches but focus on the significantly more complex task of urban driving. Theoretically, our work is motivated by [30], as we employ a similar principle of generating synthetic corrections to simulate querying an expert. Due to this, identical proven guarantees hold for our method, namely the ideal linear regret bound, mitigating the problem of covariate shift. Adversarial Imitation Learning comprises another important field [31, 32, 33], but, to the best of our knowledge, has seen little application to autonomous driving and no actual SDV deployment yet.

Neural Motion Planners are another approach used for autonomous driving. In [34] raw sensory input and HD-maps are used to estimate cost volumes of the goodness of possible future SDV positions. Based on these cost volumes, trajectories can be sampled and the lowest-cost one is selected to be executed. This was further improved in [35], where the dependency on HD-maps was dropped. To the best of our knowledge, these promising methods have not yet been demonstrated to drive a car in the real-world though.

Mid-representations and the availability of large-scale real-world AD datasets [6, 7] have been major enablers in recent years for tackling complex urban scenarios. Instead of learning policies directly from sensor data, the input of the model comprises the output of the perception system as well as an HD map of the area. This representation compactly captures the nuance of urban scenarios and allows large-scale training on hundreds or thousands of hours of real driving situations. This led to new state-of-the-art solutions for motion forecasting [8, 9]. Moreover, [10] demonstrated that using mid-representations, large-scale datasets and simple behavioral cloning with perturbations [36] can scale and learn robust planning policies. The difficulty of this approach, however, is in engineering the noise model to overcome the covariate shift problem. In our work we are inspired by this approach, but attempt to learn robust policies using policy gradient optimisation [37] featuring unrolling and evaluating the policy during training. This implicitly avoids the problem of covariate shift and leads to superior results. This approach is, however, more computationally expensive and requires a simulator. To solve this, we show how a fast and powerful simulator can be constructed directly from real-world logs enabling scalability of this approach.

Data-driven simulation. A realistic simulator is useful for both training and validation of ML models. However, many current simulators (e.g. [19, 38]) depend on heuristics for vehicle control and do not capture the diversity of real world behaviours. Data-driven simulators are designed to alleviate this problem. [23] created a photo-realistic simulator for training an end-to-end RL policy. [5] simulated a bird's-eye view of dense traffic on a highway. Finally, two recent works [39, 40] developed data-driven simulators and showed their usefulness for training and validating ML planners. In this work we show that a simpler, differentiable simulator based on replaying logs is effective for training.

3 Differentiable Traffic Simulator from Real-world Driving Data

In this section we describe a differentiable simulator S that approximates new driving experiences based on an experience $\bar{\tau}$ collected in the real world. This simulator is used during policy learning for the closed-loop evaluation of the current policy's performance and computing the policy gradient. As shown in Section 5, differentiability is an important building block for achieving good results, especially when employing auxiliary costs.

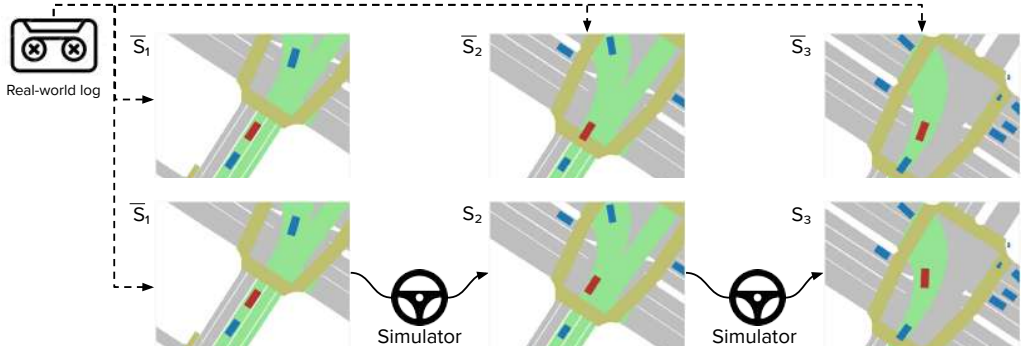


Figure 2: Differentiable simulator from observed real-world logs: based on a ground truth log (top row), we unroll a new trajectory corresponding to different SDV actions (e.g. given by a planner) in the simulator approximating the vectorized world representation (bottom row)

We represent the real-world experience $\bar{\tau}$ as a sequence of state observations $\bar{s}_t$ around the vehicle over time:

$$\bar{\tau} = \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_T\}. \tag{1}$$

We use a vectorized representation based on [8], in which each state observation $\bar{s}_t$ consists of a collection of static and dynamic elements $e_t^1, e_t^2, ..., e_t^K$ around the vehicle pose $\bar{p}_t \in SE_2$, with SE_2 denoting the special Euclidean group. Static elements include traffic lanes, stop signs and pedestrian crossings. These elements are extracted from the underlying HD semantic map of the area using the localisation system. The dynamic elements include traffic lights status and traffic participants (other cars, buses, pedestrians and cyclists). These are detected in real-time using the on-board perception system. Each element e_t^j includes a pose $q_t^j \in SE_2$ relative to the SDV pose p_t, as well as additional features, such as the element type, time of observation, and other optional attributes, e.g. the color of associated traffic lights, recent history of moving vehicles, etc. The full details of this representation are provided in Appendix C.

Goal of the simulation is to iteratively generate a sequence of state observations $\tau = \{s_1, s_2, \ldots, s_T\}$ that corresponds to a different sequence of driver actions $a_1, a_2, ..., a_N$ in the scenario. This is done by first computing the corresponding SDV trajectory $p_1, p_2, ..., p_N$ and then locally transforming states $\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_N$.

Updated poses of the SDV are determined by a kinematic model $p_{t+1} = f(p_t, a_t)$, which is assumed to be differentiable. The state observation s_t is then obtained by computing the new position q_t^j for each state element e_t^j using a transformation along the differences of the original and updated pose:

$$q_t^j = \bar{q}_t^j (p_t - \bar{p}_t). \tag{2}$$

See Figure 2 for an illustrative example. It is worth noting that this approximation is effective if the distance between the original and generated SDV pose is not too large.

We denote performing these steps in sequence with the step-by-step simulation transition function $s_{t+1} = S(s_t, a_t)$. Moreover, since both Equation (2) and vehicle dynamics f are fully differentiable, we can compute gradients with respect to both the state (S_s) and action (S_a). This is critical for the efficient computation of policy gradients using backpropagation through time as described in the next section.

4 Imitation Learning Using a Differentiable Simulator

In this part, we detail how we use the simulator S described in the previous section to learn a deterministic policy π to drive a car using closed-loop policy learning.

We frame the imitation learning problem as minimisation of the L1 pose distance $L(s_t, a_t) = \|\bar{p}_t - p_t\|_1$ between the expert and learner on a sequence of collected real-world demonstrations $\bar{\tau}_1, \bar{\tau}_2, ..., \bar{\tau}_N \sim \pi_E$. Note that with a slight abuse of notation we use the poses $\bar{p}_t, p_t$ here to refer to 3D vectors (x, y, θ), instead of roto-translation matrices in SE_2 – yielding the common L1 norm

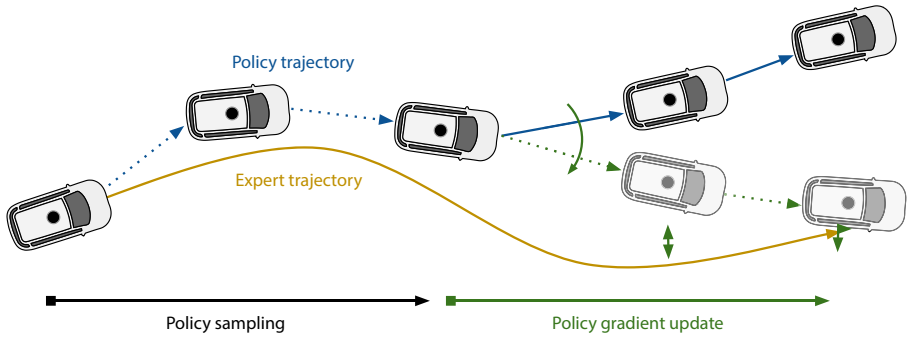


Figure 3: One iteration of policy gradient update. Given a real-world expert trajectory $\bar{\tau}$ we sample a policy state s_t by unrolling the policy π for T steps. We then compute optimal policy update by backpropagation through time.

and loss. This can be expressed as a discounted cumulative expected loss [41] on the set of collected expert scenarios:

$$J(\pi) = \mathbb{E}_{\bar{\tau}\sim\pi_E}\mathbb{E}_{\tau\sim\pi}\sum_t \gamma^t L(s_t, a_t). \tag{3}$$

Optimising this objective pushes the trajectory taken by the learned policy as close as possible to the one of the expert, as well as limiting the trajectory to the region where the approximation given by the simulator is effective. In Appendix B we further extend this to include auxiliary cost functions with the aim of optimising additional objectives.

We can use any policy optimisation method [42, 43] to optimize Equation (3). However, given that the transition $S(s_t, a_t)$ is differentiable, we can exploit it for a more effective training that does not require a separate estimation of a value function. As shown in [31, 37, 44], this results into an order of magnitude more efficient training. The optimisation process consists of repeatedly sampling pairs of expert and policy trajectories $\bar{\tau}_i$, τ_i and computing the policy gradient J_θ for these samples to minimize Equation (3). We describe both steps in detail in the following subsections.

4.1 Sampling from a Policy Distribution π

In this subsection we detail sampling pairs of expert ($\bar{\tau}$) and corresponding policy trajectory (τ) drawn from policy π.

Sampling expert trajectories $\bar{\tau}$ consists of simply sampling from the collected dataset of expert demonstrations. To generate the policy sample τ we acquire an expert state $\bar{s}_1 \in \bar{\tau}$, and then unroll the current policy π for T steps using the simulator S.

This naive method, however, introduces bias, as the initial state of the trajectory is always drawn from the expert π_E and not from the policy distribution π. As shown in Appendix B, this results in the under-performance of the method. To remove this bias we discard the first K timesteps from both trajectories and use only the remaining $T - K$ timesteps to estimate the policy gradient J_θ as described next (see Figure 3 for a visualization).

Algorithm 1: Imitation learning from expert demonstrations

Input: Expert policy samples
$\bar{\tau}_1, \bar{\tau}_2, ..., \bar{\tau}_N \sim \pi_E$
Output: Learned policy π
π = random ;
for $\bar{\tau} \sim \pi_E$ **do**
  for *t = 1 to T* **do**
    $a_t = \pi(s_t)$;
    $s_{t+1} = S(s_t, a_t)$;
  end
  $J_s^{T+1} = 0$;
  $J_\theta^{T+1} = 0$;
  for $t = T$ ***downto*** K **do**
    $J_s^t = L_s + L_a\pi_s + \gamma J_\theta^{t+1}(S_s + S_a\pi_s)$;
    $J_\theta^t = L_a\pi_\theta + \gamma(J_s^{t+1}S_a\pi_\theta + J_\theta^{t+1})$;
  end
  π = gradient_update(π, J_θ^K);
end

4.2 Computing Policy Gradient J_θ

Here we describe the computation of the policy gradient J_θ around the rollout trajectory $\tau = s_1, a_1, s_2, a_2, \ldots, s_T, a_T$ given by the current policy. This gradient can be computed for deterministic policies π using backpropagation through time leveraging the differentiability of the simulator S. Note that we denote partial differentiation with subscripts, i.e. $g_x \triangleq \partial g(x, \ldots)/\partial(x)$. We follow the formulation in [37] and express the gradient by a pair of recursive formulas:

$$J_s^t = L_s + L_a \pi_s + \gamma J_\theta^{t+1}(S_s + S_a \pi_s), \tag{4}$$

$$J_\theta^t = L_a \pi_\theta + \gamma (J_s^{t+1} S_a \pi_\theta + J_\theta^{t+1}). \tag{5}$$

The resulting algorithm is outlined in Algorithm 1 and illustrated in Figure 3. It can be implemented simply as one forward pass of length T and one backward pass of length $T - K$. To compute the policy gradient we use equations (4) and (5) recursively from $t = T$ to $t = K$ and use it to update policy parameters θ.

5 Experiments

In this section we evaluate our proposed method and benchmark it against existing state-of-the-art systems. In particular, we are interested in: its ability to learn robust policies dealing with various situations observed in the real world; its ability to tailor performance using auxiliary costs; the sensitivity of key hyper-parameters; and the impact on performance with increasing amounts of training data. Additional results can be found in the appendix and the accompanied video.

5.1 Dataset

For training and testing our models we use the Lyft Motion Prediction Dataset [6]. This dataset was recorded by a fleet of Autonomous Vehicles and contains samples of real-world driving on a complex, urban route in Palo Alto, California. The dataset captures various real-world situations, such as driving in multi-lane traffic, taking turns, interactions with vehicles at intersections, etc. Data was preprocessed by a perception system, yielding the precise position of nearby vehicles, cyclists and pedestrians over time. In addition, a high-definition map provides locations of lane markings, crosswalks and traffic lights. All models are trained on a 100h subset, and tested on 25h. The training dataset is identical to the publicly available one, whereas for the sake of execution speed for testing we use a random, but fixed, subset of the listed test dataset, which is roughly $\frac{1}{4}$ in size.

5.2 Baselines

We compare our proposed algorithm against three state-of-the-art baselines:

- Naive Behavioral Cloning (*BC*): we implement standard behavioral cloning using our vectorized backbone architecture. We do not use the SDV's history as an input to the model to avoid causal confusion (compare [10]).
- Behavioral Cloning + Perturbations (*BC-perturb*): we re-implement a vectorized version of ChauffeurNet [10] using our backbone network. As in the original paper, we add noise in the form of perturbations during training, but do not employ any auxiliary losses. We test two versions: without the SDV's history, and using the SDV's history equipped with history dropout.
- Multi-step Prediction (*MS Prediction*): we apply the meta-learning framework proposed in [30] to train our vectorized network. We observe that a version of this algorithm can conveniently be expressed within our framework; we obtain it by explicitly detaching gradients between steps (i.e. ignoring the full differentiability of our simulation environment). Differently from the original work [30], we do not save past unrolls as new dataset samples over time.

5.3 Implementation

Inspired by [8, 45], we use a graph neural network for parametrizing our policy. It combines a PointNet-like architecture for local inputs processing followed by an attention mechanism for global

Configuration		Collisions			Imitation			
Model	SDV history	Front	Side	Rear	Off-road	L2	Comfort	**I1K**
BC		79 ± 23	395 ± 170	997 ± 74	1618 ± 459	1.57 ± 0.27	**93K** $\pm$ 3K	$3{,}091 \pm 601$
BC-perturb		16 ± 2	56 ± 6	411 ± 146	82 ± 11	0.74 ± 0.01	203K $\pm$ 6K	567 ± 128
BC-perturb	✓	**14** ± 4	73 ± 7	678 ± 11	**77** ± 6	0.77 ± 0.01	636K $\pm$ 22K	843 ± 6
MS Prediction	✓	18 ± 6	55 ± 4	125 ± 14	141 ± 31	0.46 ± 0.02	595K $\pm$ 49K	341 ± 39
Ours	✓	15 ± 7	**46** ± 5	**101** ± 13	97 ± 6	**0.42** ± 0.00	637K $\pm$ 41K	**260** ± 9

Table 1: Normalized metrics for all baselines and our method – reporting mean and standard deviation for each as obtained from 3 runs. For all, lower is better. Our method overall yields best performance and lowest I1K.

reasoning. In contrast to [8], we use points instead of vectors. Given the set of points corresponding to each input element, we employ 3 PointNet layers to calculate a 128-dimensional feature descriptor. Subsequently, a single layer of scaled dot-product attention performs global feature aggregation, yielding the predicted trajectory. We found $K = 20$ and $T = 32$ to work well, i.e. we use 20 timesteps for the initial sampling and effectively predict 12 trajectory steps. γ is set to 0.8. In total, our model contains around 3.5 million trainable parameters, and training takes 30h on 32 Tesla V100 GPUs. For more details we refer to Appendix C.

For the vehicle kinematics model f we use an unconstrained model $p_{t+1} = p_t + a_t$ with $a_t \in SE_2$. This allows for a fair comparisons with the baselines as both BC-perturb and MS Prediction assume the possibility of arbitrary pose corrections. Other kinematics models, such as unicycle or bicycle models, could be used with our method as well.

All baseline methods share the same network backbone as ours, with model specific differences as described above – and BC and BC-perturb predicting a full T-step trajectory with a single forward, while MS Prediction and ours are calling the model T times. To ensure a fair comparison, also for MS Prediction we use our proposed sampling procedure, i.e. use the first K steps for sampling only. We train all models for 61 epochs with a learning rate of 10^{-4}, and drop it to 10^{-5} after 54 epochs. We note that we achieve best results for our proposed method by disabling dropout, and hypothesize this is related to similar issues observed for RNNs [46].

We refer the reader to Appendix B for ablations on the influence on data and sampling.

5.4 Metrics

We implement the metrics describe below to evaluate the planning performance. These capture key imitation performance, safety and comfort. In particular, we report the following, which are normalized – if applicable – per 1000 miles driven by the respective planner:

- **L2**: L2 distance to the underlying expert position in the driving log in meters.
- **Off-road events**: we report a failure if the planner deviates more than 2m laterally from the reference trajectory – this captures events such as running off-road and into opposing traffic.
- **Collisions**: collisions of the SDV with any other agent, broken down into front, side and rear collisions w.r.t. the SDV.
- **Comfort**: we monitor the absolute value of acceleration, and raise a failure should this exceed 3 m/s^2.
- **I1K**: we accumulate safety-critical failures (collisions and off-road events) into one key metric for ease of comparison, namely *Interventions per 1000 Miles* (I1K).

5.5 Imitation Results

We evaluate our method and all the baselines by unrolling the policy on 3600 sequences of 25 seconds length from the test set and measure the above metrics.

Table 1 reports performance when all methods are trained to optimize the imitation loss alone. Behavioral cloning yields a high number of trajectory errors and collisions. This is expected, as this approach is known to suffer from the issue of covariate shift [24]. Including perturbation during training dramatically improves performance as it forces the method to learn how to recover from

drifting. We further observe that MS Prediction yields comparable results for many categories, while yielding less rear collisions. We attribute this to the further reduction of covariate shift when compared to the previous methods: the training distribution is generated on-policy instead of being synthesized by adding noise. Finally, our method yields best results overall. It is worth noting that all models share a high number of comfort failures, due to the fact that they are all trained for imitation performance alone, which does not optimize for comfort, but only positional accuracy of the driven vehicle – which we address in the appendix.

5.6 In-car Testing

In addition to above stated simulation results, we further deployed our planner on SDVs in the real. For this, a Ford Fusion equipped with 7 camera, 3 LiDAR and 11 Radar sensors was employed. The sensor setup thus equals the one used for data collection, and during road-testing our perception and data-processing stack is run in real-time to generate the desired scene representation on the fly. For this, vehicles are equipped with 8 Nvidia 2080 TIs. Experiments were conducted on a private test track, including other traffic participants and reproducing challenging driving scenarios. Furthermore, this track was never shown to the network before, and thus offers valuable insights into generalization performance. Figure 5 shows our model successfully crossing a signaled intersection, for more results we refer to the appendix and our supplementary video.

6 Conclusion

In this work we have introduced a method for learning an autonomous driving policy in an urban setting, using closed-loop training, mid-level representations with a data-driven simulator and a large corpus of real world demonstrations. We show this yields good generalization and performance for complex, urban driving. In particular, it can control a real-world self-driving vehicle, yielding better driving performance than other state-of-the-art ML methods.

We believe this approach can be further extended towards production-grade real-world driving requirements of L4 and L5 systems – in particular, for improving performance in novel or rarely seen scenarios and to increase sample efficiency, allowing further scaling to millions of hours of driving.

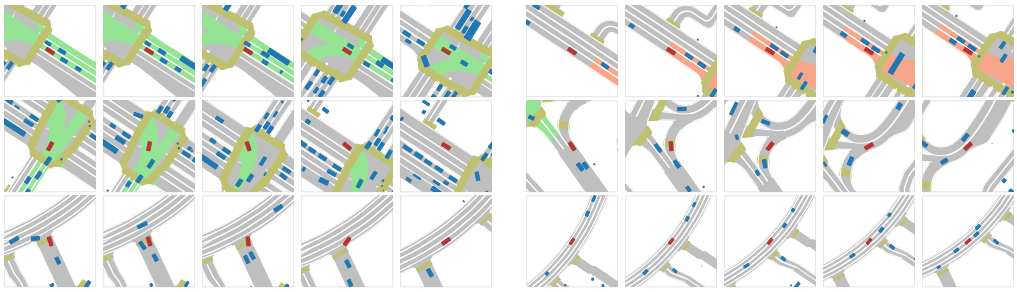


Figure 4: Qualitative results of our method controlling the SDV. Every row depicts two scenes, images are 2s apart. The SDV is drawn in red, other agents in blue and crosswalks in yellow. Traffic lights colors are projected onto the affected lanes. Best view on a screen.

Figure 5: Front-camera footage of our planner stopping for a red light, and subsequently crossing the intersection when the signal turns green (images from left to right, recorded several seconds apart).

Acknowledgments

We would like to thank everyone at Level 5 working on data-driven planning, in particular Sergey Zagoruyko, Yawei Ye, Moritz Niendorf, Jasper Friedrichs, Li Huang, Qiangui Huang, Jared Wood, Yilun Chen, Ana Ferreira, Matt Vitelli, Christian Perone, Hugo Grimmett, Parth Kothari, Stefano Pini, Valentin Irimia and Ashesh Jain. Further we would like to thank Alex Ozark, Bernard Barcela, Alex Oh, Ervin Vugdalic, Kimlyn Huynh and Faraz Abdul Shaikh for deploying our planner to SDVs in the real.

References

[1] E. D. Dickmanns. *Dynamic vision for perception and control of motion*. 2010.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[3] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[4] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong. Baidu apollo em motion planner. *ArXiv*, 2018.

[5] M. Henaff, A. Canziani, and Y. LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. *ArXiv*, 2019.

[6] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset. *Conference on Robot Learning (CoRL)*, 2020.

[7] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, P. C. De Wang, S. Lucey, D. Ramanan, and J. Hays. Argoverse: 3d tracking and forecasting with rich maps supplementary material. *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*.

[8] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[9] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun. Learning lane graph representations for motion forecasting. 2020.

[10] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. 12 2018.

[11] J. Ziegler, M. Werling, and J. Schroder. Navigating car-like robots in unstructured environments using an obstacle sensitive cost function. In *Intelligent Vehicles Symposium*, 2008.

[12] J. J. K. Jr. and S. M. Lavalle. Rrt-connect: An efficient approach to single-query path planning. In *Int. Conf. on Robotics and Automation*, 2000.

[13] T. Bandyopadhyay, K. S. Won, E. Frazzoli, D. Hsu, W. S. Lee, and D. Rus. Intention-aware motion planning. In E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, editors, *Algorithmic Foundations of Robotics X*, 2013.

[14] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun. *Junior: The Stanford Entry in the Urban Challenge*. 2009.

[15] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah. Learning to drive in a day. In *Int. Conf. on Robotics and Automation (ICRA)*, 2019.

[17] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *Transactions on Intelligent Transportation Systems*, 2021.

[18] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *ArXiv*, 2016.

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *1st Annual Conference on Robot Learning*, 2017.

[20] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. Microscopic traffic simulation using sumo. In *Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2018.

[21] Y. You, X. Pan, Z. Wang, and C. Lu. Virtual to real reinforcement learning for autonomous driving. 2017.

[22] B. Osiński, A. Jakubowski, P. Zięcina, P. Miłoś, C. Galias, S. Homoceanu, and H. Michalewski. Simulation-based reinforcement learning for real-world autonomous driving. In *Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[23] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *Robotics and Automation Letters*, 2020.

[24] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Fourteenth Int. Conf. on Artificial Intelligence and Statistics*, 2011.

[25] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *National Conference on Artificial Intelligence*, 2008.

[26] D. A. Pomerleau. *ALVINN: An Autonomous Land Vehicle in a Neural Network*. 1989.

[27] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *ArXiv*, 2016.

[28] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Micklethwaite, N. Griffiths, A. Shah, and A. Kendall. Urban driving with conditional imitation learning. In *Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[29] M. Bojarski, C. Chen, J. Daw, A. Değirmenci, J. Deri, B. Firner, B. Flepp, S. Gogri, J. Hong, L. Jackel, et al. The nvidia pilotnet experiments. *arXiv preprint arXiv:2010.08776*, 2020.

[30] A. Venkatraman, M. Hebert, and J. Bagnell. Improving multi-step prediction of learned time series models. In *AAAI*, 2015.

[31] N. Baram, O. Anschel, I. Caspi, and S. Mannor. End-to-end differentiable adversarial imitation learning. In *Int. Conf. on Machine Learning*, 2017.

[32] J. Ho and S. Ermon. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016.

[33] R. P. Bhattacharyya, B. Wulfe, D. J. Phillips, A. Kuefler, J. Morton, R. Senanayake, and M. J. Kochenderfer. Modeling human driving behavior through generative adversarial imitation learning. *ArXiv*, 2020.

[34] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. *Int. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[35] S. Casas, A. Sadat, and R. Urtasun. Mp3: A unified model to map, perceive, predict and plan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14403–14412, 2021.

[36] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *1st Annual Conference on Robot Learning*, 2017.

[37] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2015.

[38] E. Leurent. An environment for autonomous driving decision-making, 2018.

[39] L. Bergamini, Y. Ye, O. Scheel, L. Chen, C. Hu, L. Del Pero, B. Osinski, H. Grimmett, and P. Ondruska. Simnet: Learning reactive self-driving simulations from real-world observations. *Int. Conf. on Robotics and Automation*, 2021.

[40] S. Suo, S. Regalado, S. Casas, and R. Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. 2021.

[41] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, 2000.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[43] V. Konda and J. Tsitsiklis. Actor-critic algorithms. In *SIAM Journal on Control and Optimization*, 2000.

[44] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell. Deeply AggreVaTeD: Differentiable imitation learning for sequential prediction. In *Int. Conf. on Machine Learning*.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[46] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. 2014.

Urban Driver: Learning to Drive from Real-world Demonstrations Using Policy Gradients

Oliver Scheel, Luca Bergamini, Maciej Wołczyk, Błażej Osiński, Peter Ondruska
Woven Planet, Level 5
{firstname.lastname}@woven-planet.global

Appendix A: Qualitative results

Figure 1 shows our method handling diverse, complex traffic situations well - it is identical to Figure 4 of the paper, but enlarged. For more qualitative results we refer to the supplementary video.

In-car testing

In this section we report additional results of deploying our trained policy to SDVs. Figure 2 shows our planner navigating through a multitude of challenging scenarios. For more results we refer to the supplementary video - where she show additional results in the form of videos, which also contain more information, namely different camera angles, the resulting scene understanding and planned trajectory of the SDV.

Appendix B: Additional Quantitative Results

Results for Optimizing Auxiliary Costs

In this section we investigate the ability to not only imitate expert behavior, but also to directly optimize metrics of interest. This mode blends pure imitation learning with reinforcement learning and allows tailoring certain aspects of the behavior, i.e. to optimize comfort or safety. To illustrate this, we consider optimising a mixed cost function that optimizes both L1 imitation loss and auxiliary losses:

$$L_{\bar{\tau}}(s_t, a_t) = \|\bar{p}_t - p_t\|_1 + \alpha|\texttt{acc}(a_t)| + \beta \sum_{e_i \in V} \texttt{coll}(e_i, p_t) \tag{1}$$

Here $\texttt{acc}(a_t)$ is the magnitude of the acceleration at time t and $\texttt{coll}(e_i, p_t)$ is a differentiable collision indicator, with V denoting the set of other vehicles. This loss is based on [1], more details can be found in Appendix D. α, β allow to weigh the influence of these different losses.

The ability to succeed on this task requires optimally trading-off short- and long-term performance between pure imitation and other goals. Tables 1 and 2 summarize performance when including acceleration and collision loss, respectively. When including the acceleration term, we note our method is the only one to successfully trade-off performance between imitation and comfort cost, thanks to its capability to directly optimize over the full distribution: while I1K slightly increases with growing α – which is expected – we can push comfort failures down to arbitrary levels. All other models fail for at least one of these metrics, and / or are insensitive to α. When including the collision loss, results are closer together. We hypothesize this is due to $\alpha = 0$, allowing one-step corrections and thus requiring less reasoning over the full time horizon.

Ablation Studies

Figure 3 shows the impact of training dataset size on performance. We see the performance of the method improving with more data. Figure 4 demonstrates the effect of different K on the performance of closed-loop training and thus demonstrates the importance of proper sampling.

5th Conference on Robot Learning (CoRL 2021), London, UK.

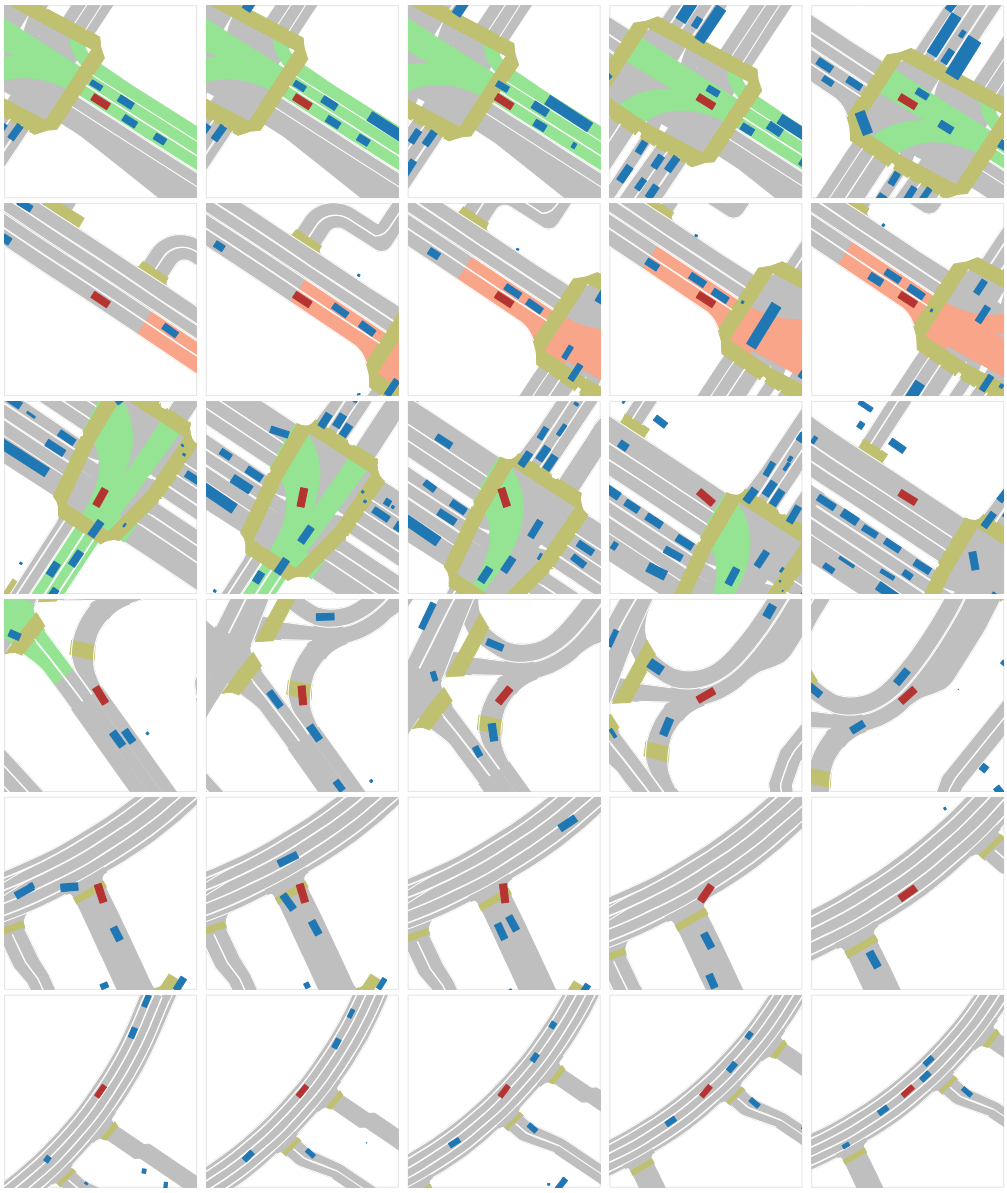


Figure 1: Qualitative results of our method controlling the SDV. Every row depicts one scene, images are 2s apart. The SDV is drawn in red, other agents in blue and crosswalks in yellow. Traffic lights colors are projected onto the affected lanes. Best view on a screen.

Discussion on Used Metrics

Metrics and their definition are naturally crucial for evaluating experiments - thus in the remainder of this section we list additional results using different thresholds and metrics. As reported in the paper, our default threshold for capturing deviations from the expert trajectory is 2m - which is based on average lane widths. Still, one can image wider lanes and less regulated traffic scenarios. Due to this Table 3 shows results of all examined methods using a threshold of 4m. Naturally, off-road failures increase, while other metrics improve due to our process of resetting after interventions. Still, one can observe that the reported results are relatively robust against such changes, i.e. the differences are small and relative trends still hold.

In the paper, for simplicity we measure comfort with one value, namely acceleration - which itself is based on differentiating speed, i.e. the travelled lateral and longitudinal distance divided by time.

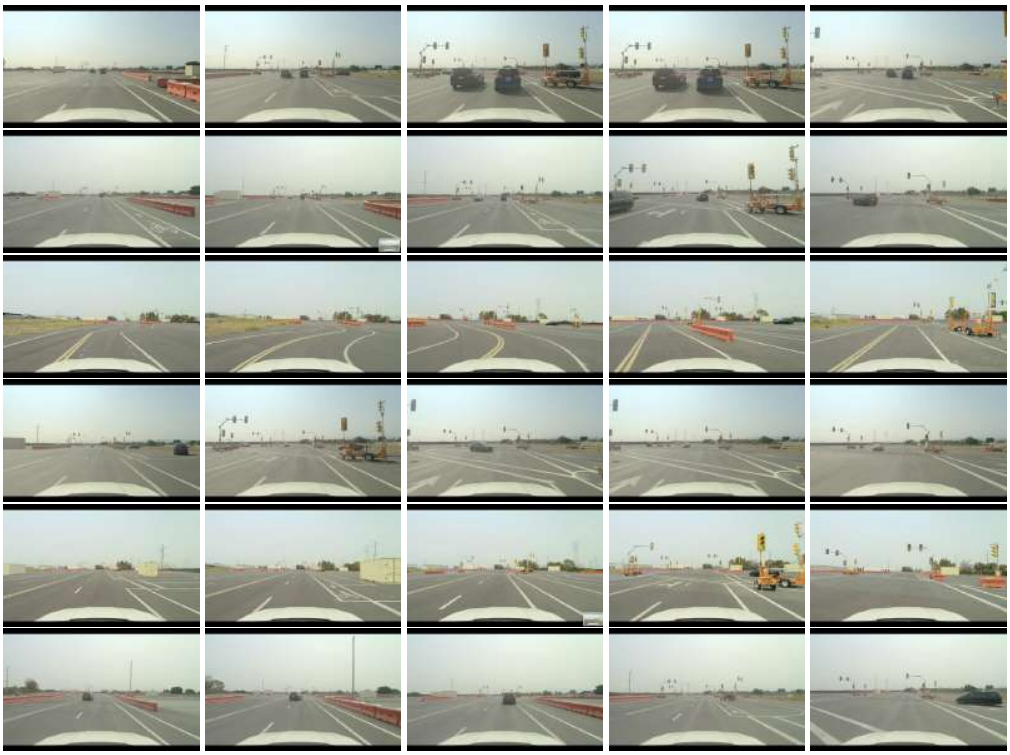


Figure 2: Qualitative results of our method controlling an SDV in the real. Every row depicts one scene, read left to right.

Configuration			Metrics	
α	β	Model	I1K	Comfort
0.01	0	BC-perturb	10,553	23,526
		MS Prediction	**1,428**	20,980
		Ours	2,512	**10,168**
0.03	0	BC-perturb	11,026	9,815
		MS Prediction	2,205	15,546
		Ours	**2,147**	**4,670**
0.1	0	BC-perturb	11,068	7,679
		MS Prediction	**2,316**	28,780
		Ours	2,737	**3,307**

Configuration			Metrics	
α	β	Model	Collisions	Comfort
0	0	BC-perturb	772	600,778
		MS Prediction	1,654	188,189
		Ours	2,055	205,131
0	1000	BC-perturb	264	858,546
		MS Prediction	612	388,632
		Ours	765	258,114
0	10000	BC-perturb	568	943,144
		MS Prediction	380	599,985
		Ours	669	508,679

Table 1: Left: influence of the acceleration term weight α. Only ours manages to find trade-offs and yields reasonable I1K and Comfort values. Right: influence of the collision term weight β. For simplicity both experiments were run with $K = 5$, note that larger K further improves performance of ours (compare Table 1 of the paper and Appendix B 2.2).

However, to reflect actual felt driving comfort, (longitudinal) jerk and lateral acceleration are better suited and more common in the industry. Therefore, Table 4 contains these additional values, and otherwise is identical to Table 1 of the original paper. These values yield more interesting insights into obtained driving behaviour, for example indicating that most discomfort is caused by longitudinal acceleration and jerk, while the lateral movement for all methods is much smoother. We further observe a similar theme as reported in the paper - namely that our method is the only one to be able to jointly optimize for performance and comfort, and that larger α yield smoother driving. Still, we note that the number of jerk failures is higher than the number of acceleration failures - which leads to promising future experiments in the form of explicitly penalizing jerk instead, or in addition to, acceleration.

To complete this excursion on metrics, we briefly discuss rear collisions. Often, they can be attributed to mistakes of other traffic participants, or non-reactive simulation (consider choosing a

Configuration			Metrics		
α	β	Model	I1K	Jerk	Lat. Acc.
0.01	0	BC-perturb	10,553	47,579	921
		MS Prediction	**1,428**	632,608	118
		Ours	2,512	621,180	128
0.03	0	BC-perturb	11,026	19,033	931
		MS Prediction	2,205	578,189	133
		Ours	**2,147**	354,341	138
0.1	0	BC-perturb	11,068	18,599	2062
		MS Prediction	**2,316**	691,540	133
		Ours	2,737	131,589	128

Configuration			Metrics		
α	β	Model	Collisions	Jerk	Lat. Acc.
0	0	BC-perturb	772	1,914,230	8,052
		MS Prediction	1,654	1,315,563	133
		Ours	2,055	1,311,728	607
0	1000	BC-perturb	264	1,922,438	29,234
		MS Prediction	612	1,455,627	1879
		Ours	765	1,935,049	261
0	10000	BC-perturb	568	1,764,526	155,852
		MS Prediction	380	1,580,696	3,131
		Ours	669	1,498,776	1,158

Table 2: Repeating Table 1, but listing (longitudinal) jerk and lateral acceleration for comfort.

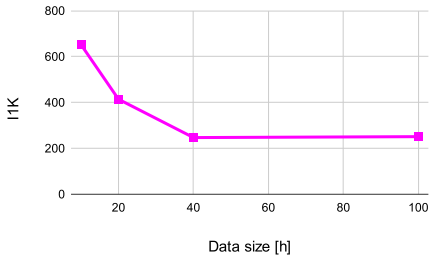


Figure 3: Influence of training data on our planner's performance: more data helps, but we seem to be reaching a plateau in performance.

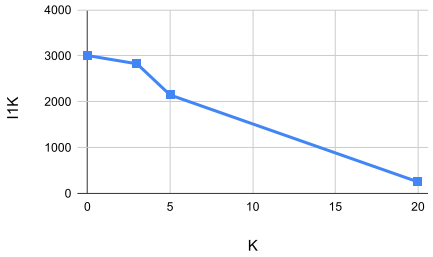


Figure 4: Importance of proper sampling: performance increases with growing K.

slightly different velocity profile, resulting in a rear collision over time). Still, rear collisions can indicate severe misbehavior, such as not starting at green traffic lights, or sudden, unsafe braking maneuvers. See Figure 5 for an example. Due to this, we do include rear collisions in our aggregation metric I1K - however note that we report all metrics separately, as well, to allow a detailed, customized performance analysis.

Appendix C: Policy architecture and state representation

In this section we disclose full details of the proposed network architecture, shown in Figure 6: Each high level object (such as an agent, lane, cross walk) is comprised of a certain number of points of feature dimension F. All points are individually embedded into a 128-dimensional space. We then add a sinusoidal embedding to points of each object to introduce an understanding of order to the model, and feed this to our PointNet implementation. This consists of 3 PointNet layers, in the end producing a descriptor of size 128 for each object. We follow this up with one layer of scaled dot-product attention: for this, the feature descriptor corresponding to ego is used as key, while all feature descriptors are taken as keys / value. We add an additional type embedding to the keys, s.t. the model can attend the values using also the object types – inspired by [2]. Via a final MLP the output is projected to the desired shape, i.e. $T \times 3$ for a trajectory of length T, in which each step is described via xy position and a yaw angle.

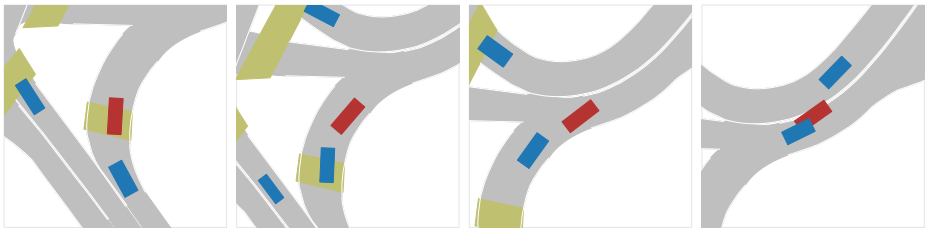


Figure 5: Showing one example of a critical rear-collision: in this case, the planner controlling the SDV (BC-perturb without ego history) decides to abruptly stop after short turn, causing the trailing car to crash into it.

Configuration		Collisions			Imitation			
Model	SDV history	Front	Side	Rear	Off-road	L2	Comfort	I1K
BC		153 ± 42	482 ± 203	1,043 ± 67	974 ± 298	8.27 ± 1.75	**102K** ± 1K	2,653 ± 483
BC-perturb		22 ± 4	57 ± 8	414 ± 142	**27** ± 5	3.06 ± 0.06	204K ± 6K	512 ± 127
BC-perturb	✓	**14** ± 6	74 ± 10	680 ± 12	**27** ± 6	3.18 ± 0.02	629K ± 23K	796 ± 12
MS Prediction	✓	22 ± 3	55 ± 3	125 ± 12	60 ± 13	2.07 ± 0.14	598K ± 49K	265 ± 17
Ours	✓	17 ± 7	**51** ± 5	**102** ± 12	40 ± 6	**1.83** ± 0.04	638K ± 41K	**210** ± 9

Table 3: Repeating Table 1 of the paper, but with a threshold of 4m for off-road failures.

Configuration		Collisions			Imitation		Comfort		
Model	SDV history	Front	Side	Rear	Off-road	L2	Jerk	Lat. Acc.	I1K
BC		79 ± 23	395 ± 170	997 ± 74	1618 ± 459	1.57 ± 0.27	958K ± 46K	71 ± 23	3,091 ± 601
BC-perturb		16 ± 2	56 ± 6	411 ± 146	82 ± 11	0.74 1,15± 0.01	1,156K ± 672K	1,115 ± 278	567 ± 128
BC-perturb	✓	**14** ± 4	73 ± 7	678 ± 11	**77** ± 6	0.77 ± 0.01	1,862K ± 46 K	7,285 ± 593	843 ± 6
MS Prediction	✓	18 ± 6	55 ± 4	125 ± 14	141 ± 31	0.46 ± 0.02	1,600K ± 14K	211 ± 21	341 ± 39
Ours	✓	15 ± 7	**46** ± 5	**101** ± 13	97 ± 6	**0.42** ± 0.00	1,750K ± 196K	507 ± 321	**260** ± 9

Table 4: Repeating Table 1 of the paper, but listing more fine-grained comfort metrics, namely (longitudinal) jerk and lateral acceleration.

A full description of our model input state is included in Table 5. We define the state as the whole set of static and dynamic elements the model receive as input. Each element is composed of a variable number of points, which can represent both time (e.g. for agents) and space (e.g. for lanes). The number of features per point depends on the element type. We pad all features to a fixed size F to ensure they can share the first fully connected layer. We include all elements up to the listed maximal number in a circular FOV of radius 35m around the SDV. Note that for performance and simplicity we only execute this query once, and then unroll within this world state.

Appendix D: Differentiable Collision Loss

We use a similar differentiable collision loss as proposed in [1]: idea is approximating each vehicle via $N = 3$ circles, and checking these for intersections. Assume loss calculation for timesteps $T-K$ to T, we then define our collision loss as:

$$\sum_{e_i \in V} \texttt{coll}(e_i, p_t) = \sum_{e_i \in V} \sum_{t=K}^{T} \texttt{pair}(e_i, p_t) \tag{2}$$

Here, $\texttt{pair}(e_i, p_t)$ describes a pair-wise collision term between our SDV and vehicle e_i at timestep t. Assume, e_i and SDV (given by pose p_t) are represented via circles C_i and C_{SDV}, then $\texttt{pair}(e_i, p_t)$

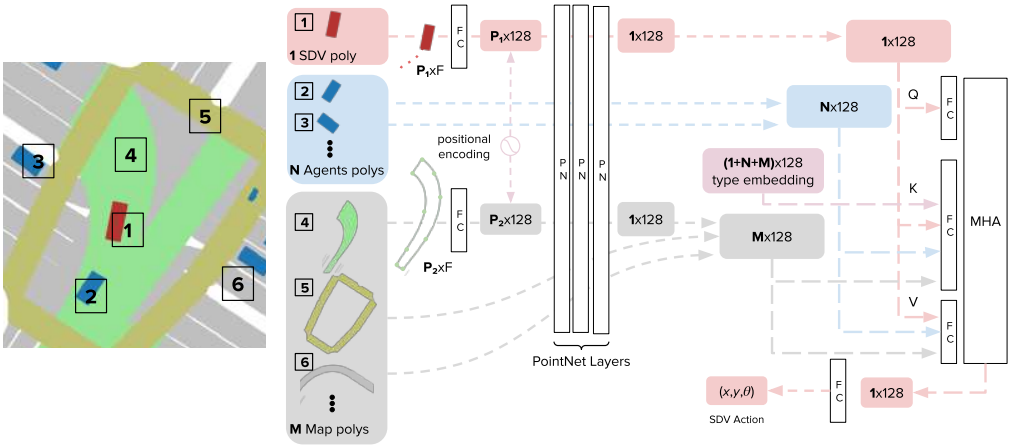


Figure 6: Overview of our policy model. Each element in the state is independently forwarded to a set of PointNet layers. The resulting features go through a Multi-Head Attention layer which takes into account their relations to output the final action for the SDV. The bird's-eye-view image on the left is only for illustrative purposes; we do not employ any rasterizations in our pipeline.

State element(s)	Elements per state	Points per element	Point features description
SDV	1	4	SDV X, Y, yaw pose of the current time step and in recent past
Agents	up to 30	4	other agents X, Y, yaw poses of the current time step and in recent past
Lanes mid	up to 30	20	interpolated X,Y points of the mid lanes with optional traffic light signal
Lanes left	up to 30	20	interpolated X,Y points of the left lanes boundaries
Lanes right	up to 30	20	interpolated X,Y points of the right lanes boundaries
Crosswalks	up to 20	up to 20	crosswalks polygons boundaries X,Y points

Table 5: Model input state description. The state is composed of multiple elements (e.g. agents and lanes) and each of them has multiple points according to its type. Each point is a multi-dimensional feature vector.

is calculated as the maximum intersection of any two such circles:

$$\texttt{pair}(e_i, p_t) = \texttt{max}_{c_i \in C_i, c_s \in C_{SDV}} \texttt{overlap}(c_i, c_s) \tag{3}$$

with

$$\texttt{overlap}(c_i, c_s) = \begin{cases} 1 - \frac{d}{r_{c_i} + r_{c_s}}, \text{ if } d \leq r_{c_i} + r_{c_s} \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

in which d denotes the distance between the respective circles' centers and r their radius. Thus, this term is 0 when the circles do not intersect, and otherwise grows linearly to a maximum value of 1.

References

[1] S. Suo, S. Regalado, S. Casas, and R. Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. 2021.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. *End-to-End Object Detection with Transformers*. 2020.

2022 IEEE International Conference on Robotics and Automation (ICRA)
May 23-27, 2022. Philadelphia, PA, USA

SafetyNet: Safe planning for real-world self-driving vehicles using machine-learned policies

Matt Vitelli*, Yan Chang*, Yawei Ye*, Ana Ferreira, Maciej Wołczyk, Błażej Osiński,
Moritz Niendorf, Hugo Grimmett, Qiangui Huang, Ashesh Jain, Peter Ondruska+

***Abstract*— In this paper we present the first safe system for full control of self-driving vehicles trained from human demonstrations and deployed in challenging, real-world, urban environments. Current industry-standard solutions use rule-based systems for planning. Although they perform reasonably well in common scenarios, the engineering complexity renders this approach incompatible with human-level performance. On the other hand, the performance of machine-learned (ML) planning solutions can be improved by simply adding more exemplar data. However, ML methods cannot offer safety guarantees and sometimes behave unpredictably. To combat this, our approach uses a simple yet effective rule-based fallback layer that performs sanity checks on an ML planner's decisions (e.g. avoiding collision, assuring physical feasibility). This allows us to leverage ML to handle complex situations while still assuring the safety, reducing ML planner-only collisions by 95%. We train our ML planner on 300 hours of expert driving demonstrations using imitation learning and deploy it along with the fallback layer in downtown San Francisco, where it takes complete control of a real vehicle and navigates a wide variety of challenging urban driving scenarios.**

I. Introduction

Self-Driving Vehicles (SDVs) have the promise to revolutionize several industries including people and goods transportation. However, the development of L4+ SDVs has proved to be a significant challenge. Today, the main bottleneck is the vehicle's ability to safely handle the 'long tail' of driving events [1]. World-class SDVs can handle common situations, but can behave unsafely in the many, rarely-occurring scenarios that are encountered on the road.

In the self-driving stack, the *planning* module is most responsible for this bottleneck. It determines what the SDV should do in any given situation. A traditional *rule-based* planning approach selects a trajectory that minimizes a hand-engineered cost function, often through classical optimization-based approaches. In order to improve its performance, engineers must design new terms in that cost function or re-tune their respective weights, for each driving scenario. This process is expensive and scales poorly to new geographies. Unlike perception, planning has benefited little from modern machine learning techniques, which leverage large quantities of data in order to avoid the hand-engineering of rules. Recently, the work of [2] has demonstrated the first machine-learned policies for autonomous driving learned directly from human demonstrations. These approaches, although they scale much better than the hand-engineering method, do not provide the interpretability and safety guarantees required to safely deploy these systems in production.

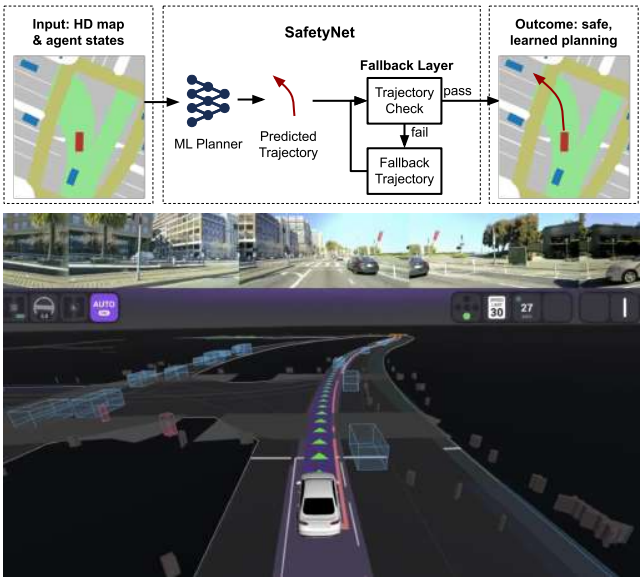


Fig. 1. *Top:* SafetyNet is the best-of-both combination of ML planning that improves with data, and rule-based safety and legality guarantees from the fallback layer. *Bottom:* an example of SafetyNet deployed to control a real-world self-driving vehicle on the streets of San Francisco.

In this work we propose SafetyNet: the first autonomous driving system to combine the strengths of an ML planner with the interpretable safety of a lightweight rule-based system, road-tested in busy San Francisco. The ML component is a high-capacity planning policy trained from expert demonstrations, and its performance scales with the amount of training data without the need for costly behavior engineering. To improve system safety, decisions of the ML planner pass through a lightweight *fallback layer*: a simple, rule-based system that tests the decisions against a small set of checks, and can minimally modify them to improve safety if required. This allows SafetyNet to transparently enforce safety and legality constraints.

This combination outperforms ML-only systems and allows us to *safely* deploy an ML planner in the busy streets of San Francisco, constituting the first demonstration of its kind. Our system exhibits a variety of maneuvers such as lane-following, keeping the distance to other vehicles, and

* Equal contribution.
+ The research was conducted at Woven Planet, Level 5. Correspondence to moritz.niendorf@woven-planet.global.
Videos are available at `safety.l5kit.org`

2022 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9681-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICRA46639.2022.9811576

navigating intersections.

Our key contributions are:

- A novel combination of machine learning and a lightweight hand-engineered system to control a self-driving vehicle that learns from data while offering safety and legality guarantees.
- The first evaluation of such a system in the challenging, real-world, urban environment of downtown San Francisco.

II. Related works

Trajectory optimization-based planning. Traditional trajectory optimization-based planning systems are widely used in both academia and industry [3]–[6]. Here the motion planning task is formulated as an optimization problem, usually by hand-engineering a cost function. The optimal trajectory is then generated by minimizing this cost using optimization algorithms, such as A* search-based methods [4], [7], sampling-based methods [8]–[10], dynamic programming [6] or combinations thereof that decompose the problem in a hierarchical fashion [11].

However, it is very difficult to hand-craft an objective function that provides a human-like trade-off between comfort, safety, and route progress over a wide variety of driving situations [12]. In comparison, encoding certain hard constraints, e.g. obstacles avoidance, physical feasibility, requires far less engineering. Building a lightweight hand-engineered system whose only purpose is to detect and correct infeasible trajectories is much simpler and more scalable.

Additionally, these hand-engineered approaches do not improve with data, and their performance does not generalize well in highly unstructured urban scenarios. Therefore, tremendous engineering efforts are needed to fine-tune them, in particular when expanding to a new operational design domain (ODD) or new geographies.

Machine-learned planning. Recently, ML planning has gained attention due to successes in deep learning. This approach has the advantage of avoiding hand-crafted rules and scales well with data, thus performing better and better as more data is used for training. Therefore, this approach has great potential to handle a wide variety of driving situations [2], [13]–[15]. Next, we introduce the two most commonly used ML paradigms for motion planning.

(1) Imitation learning (IL). IL is a supervised learning approach in which a model is trained to mimic expert behavior. The first application of IL to autonomous driving was the seminal ALVINN [16] back in 1989, which mapped the sensor data to steering and performed rural road following. More recently, [13], [15] demonstrated end-to-end driving using multiple-camera input alone, but the real-world driving results are limited to simple tasks such as lane follow or urban driving with light traffic. ChauffeurNet [2] proposed to apply IL on a bird's eye view of a scene and use synthetic perturbations to alleviate the covariate shift problem [17], but it is yet to be tested in real-world urban environments. Promising contributions to the agent prediction problem [18]–[20] have the potential to be used for the planning problem, but their performance in a closed-loop setting is not evaluated.

(2) RL & IRL. Reinforcement learning (RL) is well-suited for sequential decision processes such as self-driving as it handles the interaction between the agent and the environment. Several methods [21]–[23] have been proposed to apply RL to autonomous driving. In particular [21] proposes combining learned and rule-based components, similarly to us, but the reported results are only from simulation. On the other hand, inverse reinforcement learning (IRL) [24], [25] is another popular ML paradigm applied to autonomous driving, which infers the underlying reward function based on expert demonstrations as well as a model of the environment. However, all these methods are yet to be evaluated in real-world urban driving.

The ML planning approaches introduced above, although very promising, do not provide safety guarantees, which prevent them to be deployed at scale in the real world. We are inspired by this paradigm but aim to mitigate this limitation by the SafetyNet proposed in this paper.

Hybrid approaches. The combination of ML and traditional motion planning techniques falls mostly into two categories: *ML-based heuristics*, which are leveraged to improve traditional planning algorithms, e.g. in terms of speed up [26]–[29]. *Modular approaches*, where expert planners are leveraged to generate the trajectory candidates, e.g., by evaluating trajectories against a ML-based cost volume [14], [30]. The latter of these works, [30] also provides safety guarantees based on imposing a very high cost on trajectories leading to a potential collision. These safety guarantees were not however verified in the real world.

Another specific area of research that has emerged in this field is the study of safety frameworks [31], [32]. While this work is relevant, our goal is not to propose a comprehensive framework for safety, but rather a simple yet effective method that allows for the deployment of a powerful neural network planner that learns and improves with data while ensuring certain safety and legality constraints.

SafetyNet leverages the strengths of the expert system to guarantee certain determinism, legality, and safety rules for specific scenarios while relying on the ML motion planner for nominal trajectory generation.

III. Hybrid ML planning system

In this section we describe SafetyNet, our system for combining a machine-learned motion planner with an effective fallback layer to deliver a trajectory planning system for SDVs. The approach uses the best of both worlds, an ML planner that learns to drive from expert driving demonstrations, supported by a rule-based fallback layer to provide interpretable safety constraints, such as collision avoidance and adherence to traffic rules. The fallback layer is simple and scalable because it relies on basic hand-engineered feasibility checks, and corrects infeasible ML trajectories with a simple classical trajectory generator of small scope.

The SafetyNet system is outlined in Fig. 1. It is composed of an ML neural policy network ("ML Planner") $\mathcal{M}$ that

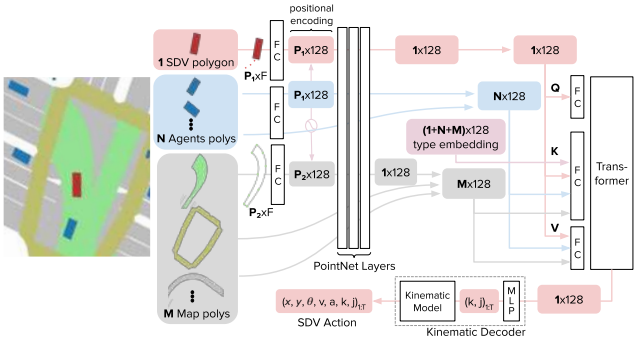


Fig. 2. The neural network architecture of the presented ML planning model is inspired by VectorNet [33]. The vectorized information on each agent and map element is encoded by a PointNet network. This local information is combined by a Transformer into global embedding. The embedding is later translated into actions via kinematic decoder.

takes as input I the environment state around the SDV and produces an intended trajectory $\bar{\tau}$ to be taken by SDV. This trajectory is validated to obey the required constraints and in the case it does not satisfy them the closest trajectory τ^i is taken from a set of safe trajectory candidates.

A. Input and Output

Input representation. The input data is encoded in an ego-centric frame of reference where the SDV is always at a fixed location relative to a frame. As shown in Fig. 2, the input to our model consists of:

1) SDV: current and past poses of the SDV and its size.
2) Agents: current and past poses of perceived agents, their sizes, and object type (e.g. vehicle, pedestrian, cyclist) produced by the SDV's perception system.
3) Static map elements: road network from High Definition (HD) maps including lanes, crosswalks, stop lines, localized using the SDV's localization system.
4) Dynamic map elements: traffic light states, and static obstacles detected by the perception system (e.g. construction zones).
5) Route: the intended global route for the SVD to follow.

We use a vectorized input representation based on [33] to featurize each element into vector sets. Each vector includes pose features - relative to the SDV pose - as well as additional features (e.g. element type, time of observation, etc). The number of feature vectors per element varies due to history availability or number of geometry points. The number of total elements in the frame varies but cannot exceed network capacity. We limit the number of elements based on a region-of-interest around the SVD.

Output representation. We define a trajectory τ as a sequence of T discrete states separated uniformly in time by Δt. Each state s_t is defined as:

$$s_t = \{x_t, y_t, \theta_t, v_t, a_t, k_t, j_t\}. \tag{1}$$

where x_t, y_t, θ_t correspond to the pose of the rear axle of the SDV w.r.t. a fixed coordinate frame at time t and v_t, a_t, k_t, j_t correspond to the velocity, longitudinal acceleration, curvature and jerk respectively.

B. ML planner

The ML planning component of our system takes the input I capturing the states around the SDV and outputs the trajectory $\bar{\tau}$ to be executed.

Model architecture. Inspired by [33], our model is built on a hierarchical graph network-based architecture. It consists of a PointNet-based [34] local subgraph for processing local information from vectorized inputs and a global graph using a Transformer encoder [35] for reasoning about interactions over agents and map features. Details of the architecture are represented in Fig. 2.

To ensure the predicted trajectories are physically feasible, we introduce a kinematic decoder, which models the vehicle kinematics using a unicycle model [5]. The decoder includes a 3-layer multilayer perceptron (MLP) that takes in the transformer encoder embedding and predicts longitudinal jerk $j_{1:T}$ and curvature $k_{1:T}$, for each time step within the prediction horizon T. Then a kinematic layer takes the predictions as well as the current ego state to roll out the next state of the ego:

$$s_{t+1} = f(s_t, k_t, j_t, \gamma), \tag{2}$$

where f is the update function of the kinematic model, γ comprises a set of parameters for vehicle kinematic constraint, including the maximum allowed jerk, acceleration, curvature and steering angle, which are used to clip controls to ensure physical feasibility. Following the deep kinematic model introduced in [36], we implement f as follows:

$$s_{t+1} = s_t + \dot{s}_t \Delta t, \tag{3}$$

where the state derivatives are computed as follows:

$$\begin{aligned} \dot{x}_t &= v_t \cos\theta_t,\ \dot{y}_t = v_t \sin\theta_t, \\ \dot{\theta}_t &= k_t v_t,\ \dot{v}_t = a_t,\ \dot{a}_t = j_t. \end{aligned} \tag{4}$$

Training framework. We use imitation learning to train a driving policy that mimics expert driving behavior by minimizing the L1 loss between the poses generated by the model and the ground truth poses. Following [2], we include perturbations to extend the distribution of states seen during the training and thus reduce the impact of the covariate shift [16], [17]. Although previous work used a pre-solver to smooth the target trajectory after applying perturbations, we can skip that thanks to the fact that we are using a kinematic decoder. Instead, we can simply penalize large values of jerk and curvature to reduce jerk and improve driving comfort. The final loss is then:

$$L = \sum_{t=1}^{T} \|p_t - \hat{p}_t\|_1 + \alpha\|k_t\|_2 + \beta\|j_t\|_2, \tag{5}$$

where p_t is the predicted pose (x_t, y_t, θ_t) at time t, $\hat{p}_t$ is the target pose, and α and β are hyperparameters.

C. Fallback Layer

After generating an ML trajectory, our system evaluates it along several dimensions for dynamic feasibility, legality, and collision probability, and determines a trajectory label

{*Feasible*, *Infeasible*}. The details of these checks are described below.

Dynamic feasibility. We evaluate whether the input trajectory remains within a feasible envelope characterized by the SDV's dynamics limits. Concretely, we evaluate each trajectory state and check whether the parameters, including longitudinal jerk, longitudinal acceleration, curvature, curvature rate, lateral acceleration, and steering jerk (curvature rate $\times$ velocity) are within reasonable bounds.

The bounds for those parameters were obtained from real-world vehicle testing. In practice, we typically use more conservative limits for jerk, longitudinal acceleration, and lateral acceleration to remain within comfortable limits.

Legality. For a given trajectory, we evaluate whether it is violating the traffic rules. A trajectory will be labeled as *Infeasible* if any of the following violations happens:

- Running the stop sign,
- Violating the right of way,
- Running a red traffic light,
- Leaving the drivable surface.

Collision likelihood. We check each ML trajectory state for collisions with the predicted poses of other agents. Agents predictions are generated by an in-house prediction module. The prediction module takes into input agents and SDV at each scene frame and can implicitly consider interactions between them. Collision detection is performed by rasterizing future agent predictions and checking for overlaps with planned ego poses. Additionally, we also check for longitudinal distance, time-to-collision, and time headway violations along the trajectory. The trajectory is labeled *Infeasible* if any of the collision likelihood checks fail.

Fallback trajectory generation. The ML trajectory is directly executed when labeled *Feasible*. If the trajectory is labeled *Infeasible*, we select a feasible fallback trajectory as close as possible to the ML trajectory.

For this we use a classical optimization-based trajectory generation method, based on [37], which generates a number of lane-aligned trajectory candidates τ^i. These candidates consist of speed keeping, distance keeping, and emergency stopping maneuvers. Our implementation can be made less conservative and expanded to other fallback behaviors of interest. However, such increase in scope for the fallback trajectory will come at a trade-off of extra engineering effort.

Each of the generated trajectories is checked for feasibility as described above and the trajectory candidate which is most similar to the ML trajectory is selected for execution:

$$\tau = \underset{\tau^i}{\operatorname{argmin}} \|\bar{\tau} - \tau^i\|_2. \tag{6}$$

For qualitative examples of how the fallback layer affects the trajectory refer to the Section IV-D.

IV. Experiments

In this section we evaluate our system across several dimensions: (a) the performance of the ML planner in simulation when trained on an increasing amount of data (no fallback layer), (b) the effectiveness of the fallback layer in simulation, and (c) the performance of SafetyNet in the real world when controlling a real vehicle in San Francisco.

A. Data

We created an in-house dataset to train and evaluate our system: 300 hours for training, 80 hours for testing - of urban driving collected in Palo Alto and San Francisco. It contains a wide range of driving scenarios in a densely populated urban environment. The dataset consists of 25s *scenes*, which capture the perception output, the SDV trajectory, and HD maps. It takes $\sim$5 hours to train the model on the full 300 hours using 64 NVIDIA V100 GPUs.

B. Metrics

We validate our planner by running a large-scale real-world driving dataset on our in-house simulator. During simulation, we execute the hybrid planner and motion controller modules, and simulate the SDV motion. The vehicle model used is calibrated to the real-world SDV. Since the simulated SDV pose can diverge significantly from the originally logged pose along the dataset scenes, we allow the other road *agents* to be reactive in their longitudinal behavior, avoiding collisions while preserving their trajectories from the dataset. The reactive agents interface is built such that the same prediction library is used in real-world and closed-loop evaluation (any prediction performance differences between logged and reactive agents is monitored in-house). Our in-house simulator is further validated to guarantee that neither collision rate nor discomfort braking are underestimated.

In order to evaluate the *closed-loop* performance of the model, each scene in the test set is replayed in simulation with the SDV following the hybrid planner output for the full duration of the scene. The following binary events are recorded:

1) *Collisions*: the simulated SDV is $<$5cm from the road boundaries, static obstacles, or agents.
2) *Close-calls*: the simulated SDV has no collision, but either gets within 25cm of another agent, has a time-to-collision $<$1.5s, or has a time headway $<$1s.
3) *Discomfort braking*: the simulated SDV's jerk drops below -5m/s^3.
4) *Passiveness*: the simulated SDV travels slower than its behavior in the dataset by < -5m/s and it is spatially behind the SDV's logged dataset position.
5) *Off road events*: the simulated SDV deviates from the dataset route center line by $>$10m.

The events recorded for each scene are aggregated and normalized by the number of miles driven in the simulations. All metrics are reported as the total number of events per 1000 miles.

To assess the ability of the planning model to fit to the dataset trajectories, we evaluate its *open-loop* performance. To this end, we compute the Average Displacement Error (ADE) between the positions from the ML trajectory output at each frame and the logged SDV's positions. Results are presented in TABLE I).

Finally, we test how the system performs in the real world by deploying it in downtown San Francisco. We evaluate the ML planner performance via how often the fallback trajectory is used and we provide qualitative examples.

Training set size	Average Displacement Error [m] @1s	@2s	@3s	@4s
5h	0.24	0.90	1.70	2.52
50h	0.14	0.51	0.98	1.53
150h	0.12	0.40	0.77	1.22
300h	0.11	0.37	0.73	1.18

TABLE I. Open-loop performance improves with the training set size.

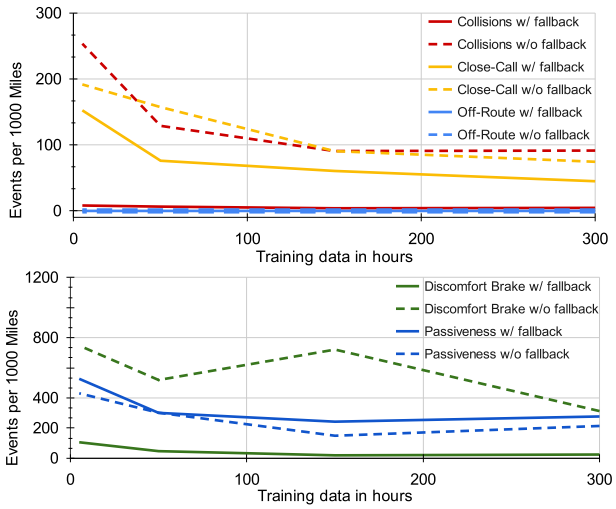


Fig. 3. Safety (*top*) and comfort (*bottom*) events as a function of dataset size for the ML planner (with and without fallback layer). The addition of the fallback layer significantly improves every metric, save passiveness.

C. Effect of dataset size

For this experiment, the fallback layer is disabled, and the ML planner's trajectory is always executed. We evaluate the performance of various ML planners trained with varying dataset sizes: {5, 50, 150, 300} hours in simulation.

Looking at the dashed lines in Fig. 3 (w/o fallback) we observe that the closed-loop performance across all safety and comfort metrics increases with training set size, and that more data should continue improving performance, albeit slowly. Models trained on <50h produce unstable driving policies that have significantly more collisions, off-route events, or even cases where the SDV remains completely still (passiveness). On the full dataset size, performance improves significantly and the learned driving policy is able to reduce collisions, close-call and discomfort brakes, and reliably follows the route. In terms of passiveness, performance plateaus, which we attribute to causal confusion inherent to open-loop training [38]. Similarly, the ADE for open-loop predictions improves with more training data (TABLE I).

D. Effect of fallback layer

We evaluate the effectiveness of the fallback layer in simulation by comparing the SafetyNet performance with, and without the fallback layer enabled. Here the ML planner is trained on the full 300h dataset. As shown in TABLE II, collisions and discomfort braking are very significantly reduced with the fallback layer enabled. This demonstrates the value of the fallback layer and its importance for real world deployment. Crucially, we see that the solid lines in Fig. 3 (w/ fallback) are close to zero regardless of the maturity of the ML planner. This indicates that we can safely deploy not only well-performing ML planners, but even immature ones (trained on <50 h of data), thus facilitating faster development and evaluation cycle.

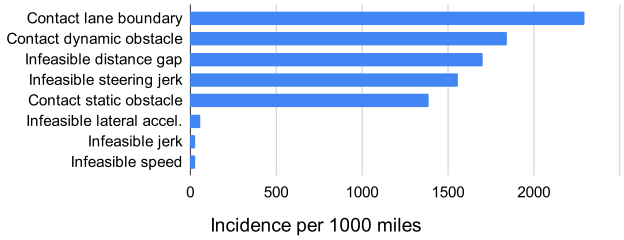


Fig. 4. The distribution of causes for a fallback trajectory to be used over the ML Planner trajectory.

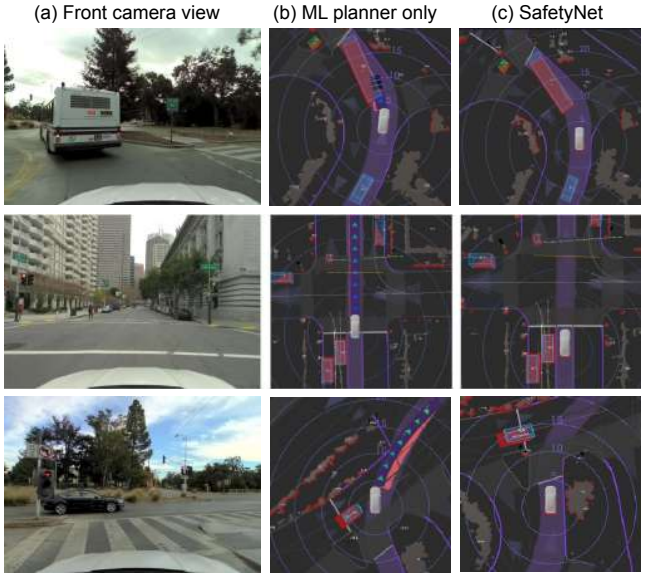


Fig. 5. Examples of when the fallback layer prevents unsafe behavior caused by ML planner. *[Top]* avoiding collision with a bus, *[middle]* running a red light, and *[bottom]* ensuring yielding to right-of-way vehicle. We report the unsafe trajectory (col. b), and SafetyNet safe trajectory (col. c).

Passiveness increases when the fallback layer is enabled, due to the SDV driving more conservatively. We see it is a necessary trade-off for reducing collisions and close calls.

In Fig. 4 we show the distribution of events that trigger the fallback trajectory to be used during simulation. The main causes for fallback behaviors are the ML trajectory leaving the drivable surface, contacting dynamic agent predictions, infeasible distance gap with other agents, infeasible steering jerk, and contact with static obstacles. By manually triaging these ML planner failure cases, we see (a) contact lane boundary issues are a result of the ML planner cutting corners too closely, and (b) collisions are caused by the network not paying enough attention to the size of the large vehicles (Fig. 5, *top*). Moreover, near traffic light intersections the ML planner occasionally generates trajectories that do not properly yield to oncoming traffic (Fig. 5, *bottom*). It is caused by the synthetic perturbations applied to the ego poses during training. Note that in all these examples, the fallback layer successfully prevented collisions from happening.

	# events per 1k miles				
Planner Type	**Collisions**	**Close Calls**	**Discomfort Braking**	**Passiveness**	**Deviation from Route**
ML Planner	91.5	74.5	312.8	213.9	0.0
ML Planner + Fallback Layer	4.6	45.0	24.2	277.0	0.0
Change Rate	-95%	-40%	-92.3%	+29.5%	0%

TABLE II. Comparing the performance of an ML-Planner-only approach vs SafetyNet, in closed-loop evaluation. This shows that SafetyNet significantly reduces collisions, close calls, and discomfort breaking, at the expense of more passiveness. See Fig. 5 for some qualitative examples.

Fig. 6. SafetyNet failure case in simulation. Two simultaneous situations - SDV driving close to dividing line, and a sudden velocity change of the oncoming vehicle - combine to reveal a limitation in the implementation.

Fig. 7. Examples of successful ML planner behavior (no fallback trajectory was triggered) in the real world. *Top:* merging, *middle:* yielding to pedestrians, and *bottom:* nudging around a parked car.

E. Failure cases and limitations

There is still a small fraction of ML planner failures not caught by SafetyNet. This is mostly because the current fallback layer implementation does not limit the SDV proximity to lane boundaries, and does not incorporate agent prediction uncertainty. In Fig. 6 the SafetyNet fails to keep a comfortable lateral distance to an oncoming vehicle.

The current fallback trajectory matching assumes that the ML trajectory is "always almost correct", which is not always the case. Better trajectory matching techniques could be explored, potentially leveraging an uncertainty output in the ML planner. However, the onus should fall on the ML planner to improve with data. A simpler fallback implementation with high issue discoverability is ideal.

F. Real world testing

Finally, we extensively tested SafetyNet with ML planner (trained on 300 hours), in the real world, in densely populated downtown San Francisco, under the supervision of human safety drivers. During the 150+ mile public road testing, the model successfully performed a wide variety of challenging maneuvers including lane-following, merging, yielding to pedestrians or nudging around parked cars (Fig. 7). At the same time, the fallback layer assured the safety of the overall system, taking over around 7.9% of the total driving time. This confirms our hypothesis that although the ML planner is able to perform complex maneuvers and drive safely for most of the time (>90%), an additional layer of safety is required in certain situations. We refer the reader to the accompanying video for a more thorough analysis. We observed a total latency of <180ms@P95 on the entire planner module running Safetynet.

V. Conclusions

We present SafetyNet, a method for combining ML planners with a rule-based system fallback layer to provide safe driving in challenging, real-world urban environments. We demonstrate very significant improvements in safety and comfort metrics compared to a purely ML-based system, both in simulation as well as in challenging San Francisco streets. This approach makes it possible to safely use learned planners in the real world, and benefit from their ability to improve with more data to handle more complex situations than their purely rule-based counterparts. We believe that teams starting to explore ML planning methods, or those with state-of-the-art ML planning solutions [2], [14], [15] would benefit from incorporating a SafetyNet system. In effect, SafetyNet may facilitate the development of ML-based planners and their wider adoption in the AV industry.

We see many exciting opportunities for further development. The fallback layer can be refined to be less conservative. Regarding the ML planner, the imitation learning approach presented is relatively simple. It can be improved by drawing from recent advancements in model-based Reinforcement Learning (RL) [39], offline RL [40], or closed-loop training with data-driven simulation [41].

VI. Acknowledgements

The authors would like to thank the following contributors for their work on Autonomy 2.0 at Level 5: Luca Bergamini, Sergey Zagoruyko, Oliver Scheel, Stefano Pini, Christian Perone, Lukas Platinsky, Jasper Friedrichs, Jared Wood, Yilun Chen and Alex Ozark.

REFERENCES

[1] A. Jain, L. D. Pero, H. Grimmett, and P. Ondruska, "Autonomy 2.0: Why is self-driving always 5 years away?" 2021.

[2] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worsts," in *Proceedings of Robotics: Science and Systems XV*, 2019.

[3] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Hähnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, and S. Thrun, "Junior: The stanford entry in the urban challenge," in *The DARPA Urban Challenge*, 2009, pp. 91–123.

[4] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer, 2009.

[5] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[6] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[7] Z. Ajanovic, B. Lacevic, B. Shyrokau, M. Stolz, and M. Horn, "Search-based optimal motion planning for automated driving," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4523–4530, 2018.

[8] J. J. K. Jr. and S. M. Lavalle, "Rrt-connect: An efficient approach to single-query path planning," in *Int. Conf. on Robotics and Automation*, 2000.

[9] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[10] T. Bandyopadhyay, K. S. Won, E. Frazzoli, D. Hsu, W. S. Lee, and D. Rus, "Intention-aware motion planning," in *Algorithmic Foundations of Robotics X*, E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, Eds., 2013.

[11] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi, "A behavioral planning framework for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 458–464.

[12] Y. Chang, S. Omari, and M. S. Vitelli, "Systems and methods for determining vehicle trajectories directly from data indicative of human-driving behavior," Jun. 10 2021, uS Patent App. 16/706,307.

[13] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[14] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun, "End-to-end interpretable neural motion planner," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[15] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Micklethwaite, N. Griffiths, A. Shah, and A. Kendall, "Urban driving with conditional imitation learning," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 251–257, 2020.

[16] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, vol. 1, 1989.

[17] S. Ross, G. J. Gordon, and J. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, 2011.

[18] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," in *Conference on Robot Learning*, 2020.

[19] H. Zhao, J. Gao, T. Lan, C. Sun, B. Sapp, B. Varadarajan, Y. Shen, Y. Shen, Y. Chai, C. Schmid *et al.*, "Tnt: Target-driven trajectory prediction," *arXiv preprint arXiv:2008.08294*, 2020.

[20] J. Ngiam, B. Caine, V. Vaudevan, Z. Zhang, H.-T. L. Zhengdong, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal, D. Weiss, B. Sapp, Z. Chen, and J. Shlens, "Scene transformer: A unified multi-task model for behavior prediction and planning," *arXiv preprint arXiv:2106.08417*, 2021.

[21] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[22] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *2007 Frontiers in the Convergence of Bioscience and Information Technologies*. IEEE, 2007, pp. 645–650.

[23] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[24] B. D. Ziebart, A. L. Maas, J. Bagnell, and A. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI*, 2008.

[25] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv: Learning*, 2015.

[26] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 2118–2124.

[27] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 2646–2652.

[28] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[29] H. Pulver, F. Eiras, L. Carozza, M. Hawasly, S. Albrecht, and S. Ramamoorthy, "Pilot: Efficient planning by imitation learning and optimisation for safe autonomous driving," *arXiv preprint arXiv:2011.00509*, 2020.

[30] S. Casas, A. Sadat, and R. Urtasun, "Mp3: A unified model to map, perceive, predict and plan," in *IEEE/CVF International Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 403–14 412.

[31] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *CoRR*, vol. abs/1708.06374, 2017. [Online]. Available: http://arxiv.org/abs/1708.06374

[32] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.

[33] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "VectorNet: Encoding HD maps and agent dynamics from vectorized representation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[34] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3d classification and segmentation," *arXiv preprint arXiv:1612.00593*, 2016.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[36] H. Cui, T. Nguyen, F.-C. Chou, T.-H. Lin, J. Schneider, D. Bradley, and N. Djuric, "Deep kinematic models for kinematically feasible vehicle trajectory predictions," 2020.

[37] M. Werling, S. Kammel, J. Ziegler, and L. Gröll, "Optimal trajectories for time-critical street scenarios using discretized terminal manifolds," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 346–359, 2012.

[38] P. de Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[39] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *arXiv preprint arXiv:2006.16712*, 2020.

[40] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.
[41] L. Bergamini, Y. Ye, O. Scheel, L. Chen, C. Hu, L. Del Pero, B. Osinski, H. Grimmett, and P. Ondruska, "Simnet: Learning reactive self-driving simulations from real-world observations," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action

Dhruv Shah[†β], **Błażej Osiński**[†βω], **Brian Ichter**[γ], **Sergey Levine**[βγ]
[β]UC Berkeley, [ω]University of Warsaw, [γ]Robotics at Google

Abstract: Goal-conditioned policies for robotic navigation can be trained on large, unannotated datasets, providing for good generalization to real-world settings. However, particularly in vision-based settings where specifying goals requires an image, this makes for an unnatural interface. Language provides a more convenient modality for communication with robots, but contemporary methods typically require expensive supervision, in the form of trajectories annotated with language descriptions. We present a system, LM-Nav, for robotic navigation that enjoys the benefits of training on unannotated large datasets of trajectories, while still providing a high-level interface to the user. Instead of utilizing a labeled instruction following dataset, we show that such a system can be constructed entirely out of pre-trained models for navigation (ViNG), image-language association (CLIP), and language modeling (GPT-3), without requiring any fine-tuning or language-annotated robot data. LM-Nav extracts landmarks names from an instruction, grounds them in the world via the image-language model, and then reaches them via the (vision-only) navigation model. We instantiate LM-Nav on a real-world mobile robot and demonstrate long-horizon navigation through complex, outdoor environments from natural language instructions.

1 Introduction

One of the central challenges in robotic learning is to enable robots to perform a wide variety of tasks on command, following high-level instructions from humans. This requires robots that can understand human instructions, and are equipped with a large repertoire of diverse behaviors to execute such instructions in the real world. Prior work on instruction following in navigation has largely focused on learning from trajectories annotated with textual instructions [1–5]. This enables understanding of textual instructions, but the cost of data annotation impedes wide adoption. On the other hand, recent work has shown that learning robust navigation is possible through goal-conditioned policies trained with self-supervision. These utilize large, unlabeled datasets to train vision-based controllers via hindsight relabeling [6–11]. They provide scalability, generalization, and robustness, but usually involve a clunky mechanism for goal specification, using locations or images. In this work, we aim to combine the strengths of both approaches, enabling a robotic navigation system to execute natural language instructions by leveraging the capabilities of pre-trained models *without any user-annotated navigational data*. Our method uses these models to construct an "interface" that humans can use to communicate desired tasks to robots. This system enjoys the impressive generalization capabilities of the pre-trained language and vision-language models, enabling the robotic system to accept complex high-level instructions.

Our main observation is that we can utilize off-the-shelf *pre-trained models* trained on large corpora of visual and language datasets — that are widely available and show great few-shot generalization capabilities — to create this interface for embodied instruction following. To achieve this, we combine the strengths of two such robot-agnostic pre-trained models with a pre-trained navigation model. We use a visual navigation model (**VNM**: ViNG [11]) to create a topological "mental map" of the environment using the robot's observations from a prior exploration of the environment. Given free-form textual instructions, we use a pre-trained large language model (**LLM**: GPT-3 [12]) to decode the instructions into a sequence of textual landmarks. We then use a vision-language model

[†] These authors contributed equally, order decided by a coin flip. Check out the project page for experiment videos, code, and a user-friendly Colab notebook that runs in your browser: sites.google.com/view/lmnav

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

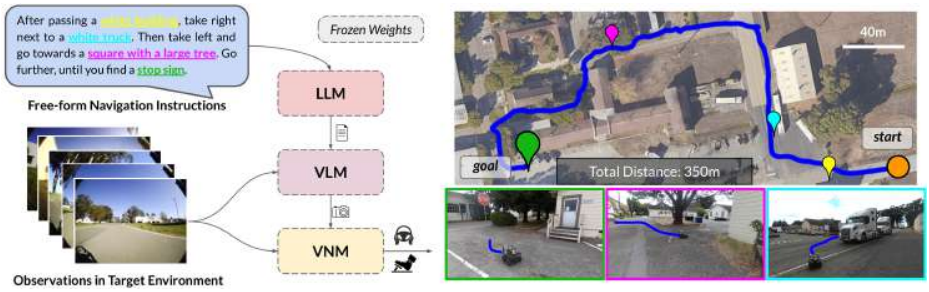


Figure 1: Embodied instruction following with LM-Nav: Our system takes as input a set of raw observations from the target environment and free-form textual instructions (left), deriving an actionable plan using three *pre-trained* models: a large language model (**LLM**) for extracting landmarks, a vision-and-language model (**VLM**) for grounding, and a visual navigation model (**VNM**) for execution. This enables LM-Nav to follow textual instructions in complex environments purely from visual observations (right) *without any fine-tuning*.

(**VLM**: CLIP [13]) for *grounding* these textual landmarks in the topological map, by inferring a joint likelihood over the landmarks and nodes. A novel search algorithm is then used to plan a path for the robot, which is then executed by **VNM**. While reducing the task of language following to a combination of grounding and subgoal selection discards a lot of useful cues such as relations and verbs, we find that it is still sufficient to follow a variety of natural language instructions.

Our primary contribution is **L**arge **M**odel **Nav**igation, or LM-Nav, an embodied instruction following system that combines three large independently pre-trained models — a robotic control model that utilizes visual observations and physical actions (**VNM**), a vision-language model that grounds images in text but has no context of embodiment (**VLM**), and a large language model that can parse and translate text but has no sense of visual grounding or embodiment (**LLM**) — to enable long-horizon instruction following in complex, real-world environments. *We present the first instantiation of a robotic system that combines the confluence of pre-trained vision-and-language models with a goal-conditioned controller, to derive actionable plans* without any fine-tuning *in the target environment.* Notably, all three models are trained on large-scale datasets, with self-supervised objectives, and used off-the-shelf with *no fine-tuning* — no human annotations of the robot navigation data are necessary to train LM-Nav. We show that LM-Nav is able to successfully follow natural language instructions in pre-explored environments over the course of 100s of meters of complex, suburban navigation, while disambiguating paths with fine-grained commands.

2 Related Work

Early works in augmenting navigation policies with natural language commands use statistical machine translation [14] to discover data-driven patterns to map free-form commands to a formal language defined by a grammar [15–19]. However, these approaches tend to operate on structured state spaces. Our work is closely inspired by methods that instead reduce this task to a sequence prediction problem [1, 20, 21]. Notably, our goal is similar to the task of VLN — leveraging fine-grained instructions to control a mobile robot solely from visual observations [1, 2].

However, most recent approaches to VLN use a large dataset of simulated trajectories — over 1M demonstrations — annotated with fine-grained language labels in indoor [1, 3–5, 22] and driving scenarios [23–28], and rely on sim-to-real transfer for deployment in simple indoor environments [29, 30]. However, this necessitates building a photo-realistic simulator resembling the target environment, which can be challenging for unstructured environments, especially for the task of outdoor navigation. Instead, LM-Nav leverages free-form textual instructions to navigate a robot in complex, outdoor environments *without* access to any simulation or any trajectory-level annotations.

Recent progress in using large-scale models of natural language and images trained on diverse data has enabled applications in a wide variety of textual [31–33], visual [13, 34–38], and embodied domains [39–44]. In the latter category, approaches either fine-tune embeddings from pre-trained models on robot data with language labels [39, 40, 44], assume that the low-level agent can execute textual instructions (without addressing control) [41], or assume access to a set of text-conditioned skills that can follow atomic textual commands [42]. All of these approaches require access to low-level skills that can follow rudimentary textual commands, necessitating language annotations for

robotic experience and a strong assumption on the robot's capabilities. In contrast, we combine these pre-trained vision and language models with pre-trained visual policies that do not use any language annotations [11, 45] *without* fine-tuning these models for the task of VLN.

Data-driven approaches to vision-based mobile robot navigation often use photorealistic simulators [46–49] or supervised data collection [50] to learn goal-reaching policies directly from raw observations. Self-supervised methods for navigation [6–11, 51] instead can use unlabeled datasets of trajectories by automatically generating labels using onboard sensors and hindsight relabeling. While such policies are adept at navigating to goal locations or images, they may be unable to parse high-level instructions such as free-form text. LM-Nav uses self-supervised policies trained in a large number of prior environments, augmented with pre-trained vision and language models for parsing natural language instructions, and deploys them in novel real-world environments *without* any fine-tuning. We emphasize that while LM-Nav relies on a pre-built topological graph, similar to prior work [11, 51, 52], this assumption may be relaxed by incorporating exploration heuristics in unseen environments [53], and can be an interesting avenue for future work.

3 Preliminaries

LM-Nav consists of three large, pre-trained models for processing language, associating images with language, and visual navigation.

Large language models are generative models of text trained on large corpora of internet text using self-supervised learning. LM-Nav uses the GPT-3 **LLM** [12] to parse instructions into a sequence of landmarks.

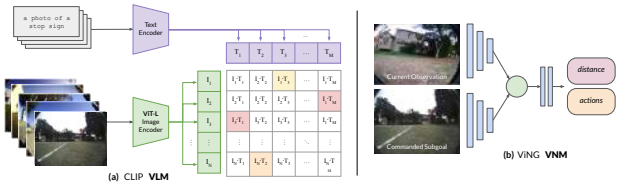


Figure 2: LM-Nav uses CLIP to infer a joint distribution over textual landmarks and image observations. **VNM** infers a goal-conditioned distance function and policy that can control the robot.

Vision-and-language models refer to models that can associate images and text, e.g. image captioning, visual question-answering, etc. [54–56]. We use the CLIP **VLM** [13], a model that jointly encodes images and text into a shared embedding space, to jointly encode a set of landmark descriptions t obtained from the **LLM** and a set of images i_k to obtain their **VLM** embeddings $\{T, I_k\}$ (see Fig. 3). Computing the cosine similarity between these embeddings, followed by a softmax operation results in probabilities $P(i_k|t)$, corresponding to the likelihood that image i_k corresponds to the string t. LM-Nav uses this probability to align landmark descriptions with images.

Visual navigation models learn navigational affordances directly from visual observations [11, 51, 57–59], associating images and actions through time. We use the ViNG **VNM** [11], a goal-conditioned model that predicts temporal distances between pairs of images and the corresponding actions to execute (see Fig. 3). The **VNM** serves two purposes: (i) given a set of observations in the target environment, the distance predictions from the **VNM** can be used to construct a topological graph $\mathcal{G}(V, E)$ that represents a "mental map" of the environment; (ii) given a "walk" (i.e., a sequence of connected subgoals to the goal), **VNM** can control the robot along this plan. The topological graph $\mathcal{G}$ is an important abstraction that allows a simple interface for planning over past experience in the environment and has been successfully used in prior work to perform long-horizon navigation [52, 53, 60]. To deduce connectivity in $\mathcal{G}$, we use a combination of learned distance estimates, temporal proximity (during data collection), and spatial proximity (using GPS measurements). For more details on the construction of this graph, see Appendix B.

4 LM-Nav: Instruction Following with Pre-Trained Models

LM-Nav combines the components discussed earlier to follow natural language instructions in the real world. The **LLM** parses free-form instructions into a list of landmarks $\bar{\ell}$ (Sec. 4.2), the **VLM** associates these landmarks with nodes in the graph by estimating the probability that each node $\bar{v}$ corresponds to each $\bar{\ell}$, $P(\bar{v}|\bar{\ell})$ (Sec. 4.3), and the **VNM** is used to infer how effectively the robot can navigate between each pair of nodes in the graph, denoted by a probability $P(\overline{v_i, v_j})$. To find the optimal "walk" on the graph that both (i) adheres to the provided instructions and (ii) minimizes traversal cost, we derive a probabilistic objective (Sec. 4.1) and show how it can be optimized using a graph search algorithm (Sec. 4.4). This walk is executed in the real world by the **VNM** model.

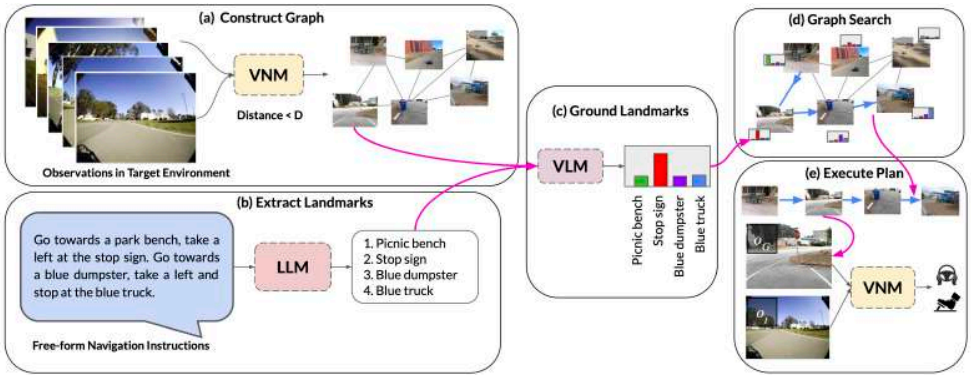


Figure 3: System overview: (a) **VNM** uses a goal-conditioned distance function to infer connectivity between the set of raw observations and constructs a topological graph. (b) **LLM** translates natural language instructions into a sequence of textual landmarks. (c) **VLM** infers a joint probability distribution over the landmark descriptions and nodes in the graph, which is used by (d) a graph search algorithm to derive the optimal walk through the graph. (e) The robot drives following the walk in the real world using the **VNM** policy.

4.1 Problem Formulation

Given a sequence of landmark descriptions $\bar{\ell} = \ell_1, \ell_2, ..., \ell_n$ extracted from the language command, our method needs to determine a sequence of waypoints $\bar{v} = v_1, v_2, ..., v_k$ to command to the robot. Typically, $k \geq n$, since each landmark needs to be visited, but the traversal might require other waypoints in between the landmarks. Finding $\bar{v}$ can formulated as a probabilistic inference problem. A key element in this formulation is access to a distribution $p(v_i|\ell_j)$ for each graph vertex v_i and landmark description ℓ_j. Recall that the graph vertices correspond to images observed by the robot, and thus, $p(v_i|\ell_i)$ represents a distribution over images given a language description. This can be obtained from the **VLM**. Intuitively, the full likelihood that we need to optimize to determine the robot's plan will now depend on two terms: likelihoods of the form $p(v_{t_i}|\ell_i)$ that describe how likely v_{t_i} is to correspond to ℓ_i for an assignment $t_1, t_2, \ldots, t_n$, and traversability likelihoods $p(\overline{v_i, v_{i+1}})$ that describe how likely is the robot to be able to reach v_{i+1} from v_i.

While we can use a variety of traversability likelihood functions, a simple choice is to use a discounted Markovian model, where the discount γ models the probability of *exiting* at each time step, leading to a termination probability of $1 - \gamma$ at each step, and a probability of reaching v_{i+1} given by $\gamma^{D(v_i, v_{i+1})}$, where $D(v_i, v_{i+1})$ is the estimated number of time steps the robot needs to travel from v_i to v_{i+1}, which is predicted by the **VNM**. While other traversability likelihoods could also be used, this choice is a convenient consequence of goal-conditioned reinforcement learning formulations [61, 62], and thus, the log-likelihood corresponds to $D(v_i, v_{i+1})$. We can use these likelihoods to derive the probability that a given sequence $\bar{v}$ can be traversed successfully, which we denote with the auxiliary Bernoulli random variable $c_{\bar{v}}$ (i.e., $c_{\bar{v}} = 1$ implies that $\bar{v}$ was traversed successfully):

$$P(c_{\bar{v}} = 1|\bar{v}) = \prod_{1 \leq i < T} P(\overline{v_i, v_{i+1}}) = \prod_{1 \leq i < T} \gamma^{D(v_i, v_{i+1})}, \tag{1}$$

The full likelihood used for planning is then given by:

$$P(\text{success}|\bar{v}, \bar{\ell}) \propto P(c_{\bar{v}} = 1|\bar{v}) P(\bar{v}|\bar{\ell}) = \prod_{1 \leq j < k} \gamma^{D(v_j, v_{j+1})} \max_{1 \leq t_1 \leq \ldots \leq t_n \leq k} \prod_{1 \leq i \leq n} P(v_{t_i}|\ell_i). \tag{2}$$

4.2 Parsing Free-Form Textual Instructions

The user specifies the route they want the robot to take using natural language, while the objective above is defined in terms of a sequence of desired landmarks. To extract this sequence from the user's natural language instruction we employ a large language model, which in our prototype is GPT-3 [12]. We used a prompt with 2 examples of correct landmarks' extractions, followed by the description to be translated by the **LLM**. Examples of instructions and landmarks extracted by the model can be found in Fig. 4. The prompt was selected to disambiguate nuanced cases, e.g. when order of landmarks in the text is different than in the expected path (see example in Fig. 4 a). For details of the *"prompt engineering"* please see Appendix A.

4.3 Visually Grounding Landmark Descriptions

As discussed in Sec. 4.1, a crucial element of selecting the walk through the graph is computing $P(v_i|\ell_j)$, the probability that landmark description v_i refers to node ℓ_j (see Eqn. 2). With each node containing an image taken during initial data collection, the probability can be computed using CLIP [13] in the way described in Sec. 3 as the retrieval task. As presented in Fig. 2, we apply CLIP to the image at node v_i and caption prompt in the form of *"This is a photo of a [ℓ_j]"*. To go from CLIP model outputs, which are logits, to probabilities we use $P(v_i|\ell_j) = \frac{\exp \text{CLIP}(v_i,\ell_j)}{\sum_{v\in V} \exp \text{CLIP}(v,\ell_j)}$. The resulting probability $P(v_i|\ell_j)$, together with the inferred edges' distances will be used to select the optimal walk.

Algorithm 1: Graph Search

Input: Landmarks $(\ell_1, \ell_2, \ldots, \ell_n)$.
Input: Graph $\mathcal{G}(V, E)$.
Input: Starting node S.
$\forall_{\substack{i=0,\ldots,n \\ v\in V}} \; Q[i, v] = -\infty$
$Q[0, S] = 0$
Dijkstra_algorithm($\mathcal{G}, Q[0, *]$)
for i in $1, 2, \ldots, n$ **do**
  $\forall v \in V Q[i, v] = Q[i - 1, v] + \text{CLIP}(v, \ell_i)$
  Dijkstra_algorithm($\mathcal{G}, Q[i, *]$)
end for
destination $= \arg\max(Q[n, *])$
return backtrack(destination, $Q[n, *]$)

4.4 Graph Search for the Optimal Walk

As described in Sec. 4.1, LM-Nav aims at finding a walk $\bar{v} = (v_1, v_2, \ldots, v_k)$ that maximizes the probability of successful execution of $\bar{v}$ that adheres to the given list of landmarks $\bar{\ell}$. We can define a function $R(\bar{v}, \bar{t})$ for a monotonically increasing sequence of indices $\bar{t} = (t_1, t_2, \ldots, t_n)$:

$$R(\bar{v}, \bar{t}) := \sum_{i=1}^{n} \text{CLIP}(v_{t_i}, \ell_i) - \alpha \sum_{j=1}^{T-1} D(v_j, v_{j+1}), \text{where } \alpha = -\log \gamma. \tag{3}$$

R has the property that $(\bar{v})$ maximizes $P(\text{success}|\bar{v}, \bar{\ell})$ defined in Eqn. 2, if and only if there exists $\bar{t}$ such that $(\bar{v}, \bar{t})$ maximizes R. In order to find such $(\bar{v}, \bar{t})$, we employ dynamic programming. In particular we define a helper function $Q(i, v)$ for $i \in \{0, 1, \ldots, n\}$, $v \in V$:

$$Q(i, v) = \max_{\substack{\bar{v}=(v_1,v_2,\ldots,v_j),v_j=v \\ \bar{t}=(t_1,t_2,\ldots,t_i)}} R(\bar{v}, \bar{t}). \tag{4}$$

$Q(i, v)$ represents the maximal value of R for a walk ending in v that visited the landmarks up to index i. The base case $Q(0, v)$ visits none of the landmarks, and its value of R is simply equal to minus the length of shortest path from the starting node S. For $i > 0$ we have:

$$Q(i, v) = \max \left(Q(i - 1, v) + \text{CLIP}(v, \ell_i), \max_{w\in\text{neighbors}(v)} Q(i, w) - \alpha \cdot D(v, w) \right). \tag{5}$$

The base case for DP is to compute $Q(0, V)$. Then, in each step of DP $i = 1, 2, \ldots, n$ we compute $Q(i, v)$. This computation resembles the Dijkstra algorithm ([63]). In each iteration, we pick the node v with the largest value of $Q(i, v)$ and update its neighbors based on the Eqn. 5. Algorithm 1 summarizes this search process. The result of this algorithm is a walk $\bar{v} = (v_1, v_2, \ldots, v_k)$ that maximizes the probability of successfully carrying out the instruction. Such a walk can be executed by **VNM**, using its action estimates to sequentially navigate to these nodes.

5 System Evaluation

We now describe our experiments deploying LM-Nav in a variety of outdoor settings to follow high-level natural language instructions with a small ground robot (Clearpath Jackal UGV platform — see Fig. 1(right) for image and Appendix C for details). For all experiments, the weights of **LLM**, **VLM**, and **VNM** are frozen — there is *no fine-tuning or annotation* in the target environment. We evaluate the complete system, as well as the individual components of LM-Nav, to understand its strengths and limitations. Our experiments demonstrate the ability of LM-Nav to follow high-level instructions, disambiguate paths, and reach goals that are up to 800m away.

Figure 4: Qualitative examples of LM-Nav in real-world environments executing textual instructions (left). The landmarks extracted by **LLM** (highlighted in text) are grounded into visual observations by **VLM** (center; overhead image not available to the robot). The resulting *walk* of the graph is executed by **VNM** (right).

5.1 Following Instructions with LM-Nav

In each evaluation environment, we first construct the graph by manually driving the robot and collecting image and GPS observations. The graph is constructed automatically using the **VNM** to predict relative distances between images in these trajectories. We tested our system on 20 queries in 2 environments, corresponding to a combined length of over 6km. The instructions include prominent landmarks that can be identified from the robot's observations, e.g., buildings and stop signs.

Fig. 4 shows qualitative examples of the path taken by the robot. In Fig. 4(a), LM-Nav is able to successfully localize the simple landmarks from its prior traversal and find a short path to the goal. While there are multiple stop signs in the environment, the objective in Eqn. 2 causes the robot to pick the correct one, minimizing overall trajectory length. Fig. 4(b) highlights LM-Nav's ability to follow complex instructions with multiple landmarks — despite the possibility of taking a shorter route directly to the final landmark, the robot follows a path that correctly visits all of the landmarks.

Missing landmarks. While LM-Nav is effective at finding a path through landmarks extracted from instructions, it relies on the assumption that the landmarks (i) exist in the environment, and (ii) can be identified by the **VLM**. Fig. 4(c) illustrates a case where the executed path fails to visit one of the landmarks — a fire hydrant — and takes a path that goes around the top of the building rather than the bottom. This failure mode is attributed to the the inability of the **VLM** to detect a fire hydrant from the robot's observations. On independently evaluating the efficacy of the **VLM** at retrieving landmarks (see Sec. 5.3), we find that despite being the best off-the-shelf model for our task, CLIP is unable to retrieve a small number of "hard" landmarks, including fire hydrants and cement mixers. In many practical cases, the robot is still successful in finding a path that visits the remaining landmarks.

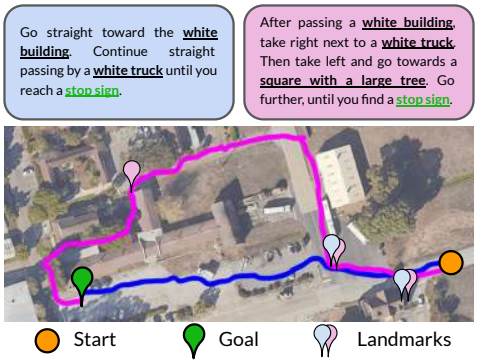


Figure 5: LM-Nav can successfully disambiguate instructions with same start-goal locations that differ slightly. The landmarks are underscored in text and their locations are marked with pins.

Disambiguation with instructions. Since the objective of LM-Nav is to follow instructions, and not merely to reach the final goal, different instructions may lead to different traversals. Fig. 5 shows an example where modifying the instruction can disambiguate multiple paths to the goal. Given the shorter prompt (blue), LM-Nav prefers the more direct path. On specifying a more fine-grained route (magenta), LM-Nav takes an alternate path that passes a different set of landmarks.

5.2 Quantitative Analysis

To quantify the performance of LM-Nav, we introduce the following metrics. A walk found by the graph search is *successful*, if (1) it matches the path intended by the user or (2) if the landmark images extracted by the search algorithm contain said landmarks (i.e. if the path visits landmarks with

System	Environment	Net Success ↑	Efficiency ↑	# Diseng. ↓	Planning ↑
GPS-Nav (No **VNM**)	`EnvSmall-10`	0.23	0.93	0.75	**0.9**
LM-Nav (Ours)	`EnvSmall-10`	**0.8**	**0.96**	**0.1**	**0.9**
	`EnvLarge-10`	**0.8**	**0.89**	**0**	**0.8**

Table 1: Quantifying navigational instruction following with LM-Nav over 20 experiments. LM-Nav can successfully plan a path to the goal, and follow it efficiently, over 100s of meters. Ablating the **VNM** (GPS-Nav) severely hurts performance due to frequent disengagements inability to reason about collisions with obstacles.

LLM Candidate	Avg. Extraction Success
Noun Chunks	0.88
fairseq-1.3B [64]	0.52
fairseq-13B [64]	0.76
GPT-J-6B [65]	0.80
GPT-NeoX-20B [66]	0.72
GPT-3 [12]	**1.0**

Table 2: GPT-3 consistently outperforms alternatives in parsing free-form instructions into landmarks.

VLM Candidate	Detection Rate
Faster-RCNN [67]	0.07
ViLD [36]	0.38
CLIP-ViT [13]	**0.87**

Table 3: CLIP-ViT produces the most reliable landmark detections from visual observations.

the same description, even if not exactly the same). *Planning success* is the fraction of successful walks found by the search algorithm. *Efficiency* of a walk is defined as the ratio of the lengths of the described route and the executed one; the value is clipped at a maximum of 1 to account for the cases when the LM-Nav executes a path shorter than the user intended. For a set of queries, we report the average efficiency over successful experiments. The *planning efficiency* is similarly defined as the ratio of the length of the described and *planned* routes. Finally, *number of disengagements* is the average number of human interventions per experiment due to unsafe maneuvers.

Table 1 summarizes the quantitative performance of the system over 20 instructions. LM-Nav generates a successful walk for 85% of them, and causes disengagement only once (an average of 1 intervention per 6.4km of traversals). Investigating the planning failure modes suggests that the most critical component of our system is the ability of **VLM** to detect certain landmarks, e.g. a fire hydrant, and in challenging lighting conditions, e.g. underexposed images.

5.3 Dissecting LM-Nav

To understand the influence of each of the components of LM-Nav, we conduct experiments to evaluate these components in isolation. For more details about these experiments, see Appendix D.

To evaluate the performance of **LLM** candidates in parsing instructions into an *ordered* list of landmarks, we compare GPT-3 (used by LM-Nav) to other state-of-the-art pre-trained language models — fairseq [64], GPT-J-6B [65], and GPT-NeoX-20B [66] — as well as a simple baseline using spaCy NLP library [68] that extracts base noun phrases, followed by filtering. In Table 2 we report the average extraction success for all the methods on the 20 prompts used in Section 5.2. GPT-3 significantly outperforms other models, owing to its superior representation capabilities and in-context learning [69]. The noun chunking performs surprisingly reliably, correctly solving many simple prompts. For further details on these experiments, see Appendix D.2.

To evaluate the **VLM**'s ability to ground these textual landmarks in visual observations, we set up an object detection experiment. Given an unlabeled image from the robot's on-board camera and a set of textual landmarks, the task is to *retrieve* the corresponding label. We run this experiment on a set of 100 images from the environments discussed earlier, and a set of 30 commonly-occurring landmarks. These landmarks are a combination of the landmarks retrieved by the **LLM** in our

	`EnvSmall-10`		`EnvLarge-10`	
Planner	Pl. Success ↑	Pl. Efficiency ↑	Pl. Success ↑	Pl. Efficiency ↑
Max Likelihood	0.6	0.69	0.2	0.17
LM-Nav (Ours)	**0.9**	**0.80**	**0.8**	**0.99**

Table 4: Ablating the search algorithm (Sec. 4.4) gives a max likelihood planner that ignores reachability information, resulting in inefficient plans that are up to 6× longer than LM-Nav for the same instruction.

experiments from Sec. 5.1 and manually curated ones. We report the detection successful if any of the top 3 predictions adhere to the contents of the image. We compare the retrieval success of our **VLM** (CLIP) with some object detection alternatives — Faster-RCNN-FPN [67, 70], a state-of-the-art object detection model pre-trained on MS-COCO [71, 72], and ViLD [36], an open-vocabulary object detector based on CLIP and Mask-RCNN [73]. To evaluate against the closed-vocabulary baseline, we modify the setup by projecting the landmarks onto the set of MS-COCO class labels. We find that CLIP outperforms baselines by a wide margin, suggesting that its visual model transfers very well to robot observations (see Table 3). Despite deriving from CLIP, ViLD struggles with detecting complex landmarks like "manhole cover" and "glass building". Faster-RCNN is unable to detect common MS-COCO objects like "traffic light", "person" and "stop sign", likely due to the on-board images being out-of-distribution for the model.

To understand the importance of the **VNM**, we run an ablation experiment of LM-Nav without the navigation model. Using GPS-based distance estimates and a naïve straight line controller between nodes of the topological graph. Table 1 summarizes these results — without **VNM**'s ability to reason about obstacles and traversability, the system frequently runs into small obstacles such as trees and curbs, resulting in failure. Fig. 6 illustrates such a case — while such a controller works well on open roads, it fails to reason about connectivity around buildings or obstacles and results in collisions with a curb, a tree, and a wall in 3 individual attempts. This illustrates that using a learned policy and distance function from the **VNM** is critical for LM-Nav to successfully navigate in complex environments.

Figure 6: GPS-Nav (red) fails to execute a plan due to its inability to reason about traversability through obstacles, while LM-Nav (blue) succeeds.

Lastly, to understand the importance of the two components of the graph search objective (Eqn. 3), we ran a set of ablations where the graph search only depends on $P(\bar{v}|\bar{\ell})$, i.e. *Max Likelihood Planning*, which only picks the most likely landmark without reasoning about topological connectivity or traversability. Table 4 shows that such a planner suffers greatly in the form of efficiency, because it does not utilize the spatial organization of nodes and their connectivity. For more details on these experiments, and qualitative examples, see Appendix D.

6 Discussion

We presented **L**arge **M**odel **Nav**igation, a robotic system that can execute textual instructions in the real-world without requiring any human annotations for navigation trajectories. LM-Nav combines three pre-trained models: the **LLM**, which parses instructions into a list of landmarks; the **VLM**, which infers joint probabilities between these landmarks and visual observations from the environment; and the **VNM**, which estimates navigational affordances (distances between landmarks) and control actions. Each model is pre-trained on its own dataset, and we show that the complete system can execute a variety of user-specified instructions in real-world environments — choosing the correct sequence of landmarks by leveraging language and spatial context — and handle mistakes (such as missing landmarks). We also analyze the impact of each pre-trained model on the full system.

Limitations and future work. The most prominent limitation of LM-Nav is its reliance on landmarks: while the user can specify any instruction they want, LM-Nav only focuses on the landmarks and disregards any verbs, propositions, adverbs, etc. (e.g., "go straight for three blocks" or "drive past the dog very slowly"), which can be lossy. Grounding such nuances is an important direction for future work. Additionally, LM-Nav uses a **VNM** that is specific to outdoor navigation with the Jackal robot, which limits wider adoption for other robot embodiments and sensor suites. An exciting direction for future work would be to swap in a "general navigation model" [74] that can be utilized broadly across robots, analogous to how the **LLM** and **VLM** handle any text or image. In its current form, LM-Nav provides a simple and attractive prototype for how pre-trained models can be combined to solve complex robotic tasks, and illustrates that these models can serve as an "interface" to robotic controllers that are trained without any language annotations. One of the implications of this result is that further progress on self-supervised robotic policies (e.g., goal-conditioned policies) can directly benefit instruction following systems. More broadly, understanding how modern pre-trained models enable effective decomposition of robotic control may enable broadly generalizable systems in the future, and we hope that LM-Nav will serve as a step in this direction.

Acknowledgments

This research was supported by DARPA Assured Autonomy, DARPA RACER, Toyota Research Institute, ARL DCIST CRA W911NF-17-2-0181, and AFOSR. BO was supported by the Fulbright Junior Research Award granted by the Polish-U.S. Fulbright Commission. We would like to thank Alexander Toshev for pivotal discussions in early stages of the project. We would also like to thank Kuan Fang, Siddharth Karamcheti, Albertyna Osińska, Vijay Badrinarayanan, and Philip J. Ball for useful discussions and feedback.

References

[1] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018. 1, 2

[2] J. Gu, E. Stefani, Q. Wu, J. Thomason, and X. Wang. Vision-and-language navigation: A survey of tasks, methods, and future directions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022. 2

[3] A. Ku, P. Anderson, R. Patel, E. Ie, and J. Baldridge. Room-Across-Room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2020. 2

[4] V. Jain, G. Magalhaes, A. Ku, A. Vaswani, E. Ie, and J. Baldridge. Stay on the path: Instruction fidelity in vision-and-language navigation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[5] A. Yan, X. E. Wang, J. Feng, L. Li, and W. Y. Wang. Cross-lingual vision-language navigation, 2019. 1, 2

[6] T. Manderson, J. C. Gamboa, S. Wapnick, J. Tremblay, H. Zhao, F. Shkurti, D. Meger, and G. Dudek. Self-supervised, goal-conditioned policies for navigation in unstructured environments. 2010. 1, 3

[7] B. Sofman, E. L. Ratliff, J. A. D. Bagnell, J. Cole, N. Vandapel, and A. T. Stentz. Improving robot navigation through self-supervised online learning. *Journal of Field Robotics: Special Issue on Machine Learning Based Robotics in Unstructured Environments*, 2006.

[8] D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[9] A. Kouris and C.-S. Bouganis. Learning to fly by myself: A self-supervised cnn-based approach for autonomous navigation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[10] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine. Self-Supervised Deep RL with Generalized Computation Graphs for Robot Navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[11] D. Shah, B. Eysenbach, G. Kahn, N. Rhinehart, and S. Levine. ViNG: Learning Open-World Navigation with Visual Goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. 1, 3

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020. 1, 3, 4, 7

[13] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021. 2, 3, 5, 7, 1

[14] P. Koehn. *Statistical Machine Translation*. Cambridge University Press, 2009. 2

[15] Y. W. Wong and R. Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, 2006. 2

[16] C. Matuszek, D. Fox, and K. Koscher. Following directions using statistical machine translation. In *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2010.

[17] D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[18] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[19] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. *Learning to Parse Natural Language Commands to a Robot Control System*. 2013. 2

[20] N. Shimizu and A. Haas. Learning to follow navigational route instructions. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, 2009. 2

[21] H. Mei, M. Bansal, and M. R. Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *AAAI*, 2016. 2

[22] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[23] H. Chen, A. Suhr, D. Misra, N. Snavely, and Y. Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[24] K. M. Hermann, M. Malinowski, P. Mirowski, A. Banki-Horvath, K. Anderson, and R. Hadsell. Learning to follow directions in street view. *CoRR*, 2019.

[25] P. Mirowski, A. Banki-Horvath, K. Anderson, D. Teplyashin, K. M. Hermann, M. Malinowski, M. K. Grimes, K. Simonyan, K. Kavukcuoglu, A. Zisserman, and R. Hadsell. The streetlearn environment and dataset. *CoRR*, 2019.

[26] A. B. Vasudevan, D. Dai, and L. Van Gool. Talk2nav: Long-range vision-and-language navigation with dual attention and spatial memory. *Int. J. Comput. Vision*, 2021.

[27] D. K. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkhin, and Y. Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. In E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[28] V. Blukis, N. Brukhim, A. Bennett, R. A. Knepper, and Y. Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning. *CoRR*, 2018. 2

[29] J. Krantz, E. Wijmans, A. Majumdar, D. Batra, and S. Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII*, 2020. 2

[30] P. Anderson, A. Shrivastava, J. Truong, A. Majumdar, D. Parikh, D. Batra, and S. Lee. Sim-to-real transfer for vision-and-language navigation. In *Proceedings of the 2020 Conference on Robot Learning*, 2021. 2

[31] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, 2019. 2

[32] R. Thoppilan, D. D. Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, Y. Li, H. Lee, H. S. Zheng, A. Ghafouri, M. Menegali, Y. Huang, M. Krikun, D. Lepikhin, J. Qin, D. Chen, Y. Xu, Z. Chen, A. Roberts, M. Bosma, Y. Zhou, C. Chang, I. Krivokon, W. Rusch, M. Pickett, K. S. Meier-Hellstern, M. R. Morris, T. Doshi, R. D. Santos, T. Duke, J. Soraker, B. Zevenbergen, V. Prabhakaran, M. Diaz, B. Hutchinson, K. Olson, A. Molina, E. Hoffman-John, J. Lee, L. Aroyo, R. Rajakumar, A. Butryna, M. Lamm, V. Kuzmina, J. Fenton, A. Cohen, R. Bernstein, R. Kurzweil, B. Aguera-Arcas, C. Cui, M. Croak, E. H. Chi, and Q. Le. Lamda: Language models for dialog applications. *CoRR*, 2022.

[33] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. *CoRR*, 2021. 2

[34] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents, 2022. 2

[35] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022.

[36] X. Gu, T.-Y. Lin, W. Kuo, and Y. Cui. Open-vocabulary object detection via vision and language knowledge distillation. In *International Conference on Learning Representations*, 2022. 7, 8

[37] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. Le, Y.-H. Sung, Z. Li, and T. Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[38] H. Song, L. Dong, W. Zhang, T. Liu, and F. Wei. CLIP models are few-shot learners: Empirical studies on VQA and visual entailment. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022. 2

[39] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021. 2

[40] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. BC-z: Zero-shot task generalization with robotic imitation learning. In *5th Annual Conference on Robot Learning*, 2021. 2

[41] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022. 2

[42] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, and M. Yan. Do as i can, not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022. 2

[43] A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, and P. Florence. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv*, 2022. 2

[44] A. Khandelwal, L. Weihs, R. Mottaghi, and A. Kembhavi. Simple but effective: Clip embeddings for embodied ai. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[45] D. Shah, B. Eysenbach, N. Rhinehart, and S. Levine. Rapid exploration for open-world navigation with latent goal models. In *5th Annual Conference on Robot Learning*, 2021. 3

[46] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra. Habitat: A Platform for Embodied AI Research. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 3

[47] F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese. Gibson env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*, 2018.

[48] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv:1712.03931*, 2017.

[49] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. AI2-THOR: an interactive 3d environment for visual AI. *CoRR*, 2017. 3

[50] A. Francis, A. Faust, H. T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T. W. E. Lee. Long-Range Indoor Navigation With PRM-RL. *IEEE Transactions on Robotics*, 2020. 3

[51] N. Hirose, F. Xia, R. Martín-Martín, A. Sadeghian, and S. Savarese. Deep visual MPC-policy learning for navigation. *IEEE Robotics and Automation Letters*, 2019. 3

[52] X. Meng, N. Ratliff, Y. Xiang, and D. Fox. Scaling Local Control to Large-Scale Topological Navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020. 3

[53] D. Shah and S. Levine. Viking: Vision-based kilometer-scale navigation with geographic hints. In *Robotics: Science and Systems (RSS)*, 2022. 3

[54] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan. Flamingo: a visual language model for few-shot learning, 2022. 3

[55] L. H. Li, M. Yatskar, D. Yin, C.-J. Hsieh, and K.-W. Chang. Visualbert: A simple and performant baseline for vision and language. In *Arxiv*, 2019.

[56] Y.-C. Chen, L. Li, L. Yu, A. E. Kholy, F. Ahmed, Z. Gan, Y. Cheng, and J. Liu. Uniter: Universal image-text representation learning. In *ECCV*, 2020. 3

[57] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-Parametric Topological Memory for Navigation. In *International Conference on Learning Representations*, 2018. 3

[58] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov. Learning to Explore using Active Neural SLAM. In *International Conference on Learning Representations (ICLR)*, 2020.

[59] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra. DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames. In *International Conference on Learning Representations (ICLR)*, 2020. 3

[60] J. Bruce, N. Sunderhauf, P. Mirowski, R. Hadsell, and M. Milford. Learning deployable navigation policies at kilometer scale from a single traversal. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, 2018. 3

[61] L. P. Kaelbling. Learning to achieve goals. In *IJCAI*, pages 1094–1099, 1993. 4

[62] K. Hartikainen, X. Geng, T. Haarnoja, and S. Levine. Dynamical Distance Learning for Semi-Supervised and Unsupervised Skill Discovery. In *International Conference on Learning Representations*, 2020. 4

[63] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1959. 5

[64] M. Artetxe, S. Bhosale, N. Goyal, T. Mihaylov, M. Ott, S. Shleifer, X. V. Lin, J. Du, S. Iyer, R. Pasunuru, et al. Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*, 2021. 7

[65] B. Wang and A. Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, 2021. 7

[66] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang, M. Pieler, U. S. Prashanth, S. Purohit, L. Reynolds, J. Tow, B. Wang, and S. Weinbach. Gpt-neox-20b: An open-source autoregressive language model, 2022. 7

[67] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 2015. 7, 8

[68] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd. spacy: Industrial-strength natural language processing in python. 2020. 7

[69] F. Rong. Extrapolating to unnatural language processing with gpt-3's in-context learning: The good, the bad, and the mysterious. http://ai.stanford.edu/blog/in-context-learning/, 2021. Accessed: 2022-06-04. 7, 1

[70] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 8

[71] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 8

[72] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 8

[73] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017. 8

[74] D. Shah, A. Sridhar, A. Bhorkar, N. Hirose, and S. Levine. GNM: A General Navigation Model to Drive Any Robot. In *arXiV*, 2022. URL https://arxiv.org/abs/2210.03370. 8