TU WIEN Informatics

Deep Off-Policy Evaluation with Autonomous Racing Cars

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Fabian Kresse, BSc
Matrikelnummer 11707724

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr. rer. nat. Radu Grosu
Mitwirkung: Dott. Mag. Luigi Berducci

Wien, 12. August 2024

Fabian Kresse | Radu Grosu

TU WIEN Informatics

Deep Off-Policy Evaluation for Autonomous Racing Cars

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Fabian Kresse, BSc
Registration Number 11707724

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr. rer. nat. Radu Grosu
Assistance: Dott. Mag. Luigi Berducci

Vienna, August 12, 2024

Fabian Kresse | Radu Grosu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Fabian Kresse, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 12. August 2024

Fabian Kresse

Danksagung

Zuallererst möchte ich mich bei Prof. Radu Grosu bedanken, der mir die Möglichkeit gegeben hat, an dieser spannenden Masterarbeit zu arbeiten. Ich schätze besonders die Bereitstellung der notwendigen Ressourcen und des unterstützenden Arbeitsumfeldes, die diese Arbeit ermöglicht haben.

Ein großer Dank gebührt auch Luigi Berducci, dessen wöchentliche Meetings und ausführliche Rückmeldungen maßgeblich zu dieser Arbeit beigetragen haben. Seine unermüdliche Unterstützung und seine Geduld beim mehrmaligen Durchlesen und Kommentieren meiner Arbeit waren von unschätzbarem Wert.

Ohne die Hilfe und das Engagement dieser beiden Personen wäre diese Arbeit nicht möglich gewesen. Vielen Dank für Ihre Unterstützung, Ihre Zeit und Ihre Expertise.

Acknowledgements

First and foremost, I would like to thank Prof. Radu Grosu for giving me the opportunity to work on this exciting thesis. I particularly appreciate the provision of the necessary resources and the supportive working environment that made this work possible.

A special thanks goes to Luigi Berducci, whose weekly meetings and detailed feedback significantly contributed to this work. His tireless support and patience in reading and commenting on my work multiple times were invaluable.

Without the help and commitment of these two individuals, this work would not have been possible. Thank you for your support, your time, and your expertise.

Kurzfassung

Roboteragenten, die mit Reinforcement Learning (RL) in einer Simulation trainiert wurden, auf die reale Welt zu übertragen, ist in der Praxis schwierig, da sich die Agenten zu stark an die Besonderheiten des Simulators anpassen. Dies ist besonders problematisch, wenn die Bewertung der Agenten im selben Simulator erfolgt, da die Überanpassung an den Simulator verborgen bleibt. Dies führt dazu, dass kostspielige Tests in der realen Welt notwendig sind, um eine genaue Leistungsabschätzung der Agenten zu erhalten.

In letzter Zeit hat Off-Policy Evaluation (OPE) vielversprechende Anzeichen gezeigt, den Bedarf an Tests in der realen Welt zumindest zu reduzieren. Dies wird dadurch erreicht, dass OPE Leistungsabschätzungen basierend auf der Verteilung der Daten in der realen Welt liefern kann. Allerdings wurde die Wirksamkeit von OPE in realen Robotikanwendungen noch nicht ausreichend untersucht. Bestehende Benchmarks basieren nur auf simulierten Daten, die die Unvorhersehbarkeit und Komplexität der realen Welt nicht abbilden. Um diese Forschungslücke zu schließen, präsentiert diese Arbeit das ersten Robotik-Benchmark für OPE-Methoden in der echten Welt. Als Roboterplattform für dieses Benchmark nutzen wir die kostengünstige und leicht zugängliche F1TENTH-Plattform.

Da die Leistung einiger bestehender OPE-Methoden unzureichend ist, wenn sie naiv auf unsere untersuchte F1TENTH-Umgebung angewendet werden, untersuchen wir spezifische Verbesserungen und führen ein Benchmarking von über 20 OPE-Methoden durch. Unter anderem stellen wir den Termination-aware Per-Decision-Weighted Importance Sampling (TPDWIS)-Schätzer vor, einen neuen Importance Sampling (IS)-Schätzer, der Trajektorien mit ungleichmäßigen Längen handhaben kann und bisherige IS-Schätzer aus der Literatur in unserem Benchmark deutlich übertrifft. Außerdem beweisen wir die Konsistenz dieses neuen Schätzers, was zeigt, dass er wahrscheinlich auf allgemeinere Probleme angewendet werden kann.

Schließlich geben wir Empfehlungen dazu ab, welche OPE-Methoden unter verschiedenen Einschränkungen verwendet werden sollten. Unser neuer Datensatz, das Benchmark, die vorgeschlagenen Verbesserungen und der neue Schätzer bieten eine solide Grundlage für die zukünftige Forschung und Entwicklung von OPE-Methoden in der realen Welt.

Abstract

Transferring robot policies trained with Reinforcement Learning (RL) from simulation to the real-world is challenging due to frequent overfitting to the simulator's particularities. This is especially problematic when policy evaluation occurs within the same simulator, as it conceals overfitting, necessitating frequent and costly real-world deployments for accurate performance estimation.

Recently Off-Policy Evaluation (OPE) has shown promise in reducing the need for extensive real-world deployment by providing performance estimates based on approximations of real-world data distributions. However, the effectiveness of OPE methods in real-world robotics has not been extensively investigated, with existing benchmarks primarily relying on simulated environments that fail to capture real-world complexities and unpredictability. Addressing this gap, this thesis introduces the first real-world robotics benchmark for OPE methods, utilizing the affordable and accessible F1TENTH platform.

As the performance of some of the existing OPE methods is inadequate when applied naively to our investigated F1TENTH environment, we explore specific improvements and benchmark over 20 OPE methods. Among these, we introduce the Termination-aware Per-Decision-Weighted Importance Sampling (TPDWIS) estimator, a novel Importance Sampling (IS) estimator capable of handling trajectories with non-uniform lengths, significantly outperforming previous IS estimators from the literature on our benchmark. Furthermore, we prove the consistency of this new estimator, indicating that it can be applied to more general environments.

Finally, we provide recommendations on the most effective OPE methods to employ under specific constraints. Our novel dataset, benchmark, proposed improvements, and new estimator offer a robust foundation for future research and development in real-world OPE methods.

Contents

CHAPTER 1

Introduction

The success of learning algorithms has revolutionized the development of intelligent machines. Reinforcement Learning (RL) algorithms, in particular, have demonstrated remarkable capabilities across a wide range of applications, from autonomous driving to robotic manipulation. These algorithms have achieved feats once thought to be far beyond the reach of automated systems, showcasing sophisticated decision-making and adaptability in complex environments.

Despite these advancements, deploying intelligent machines in real-world settings involves high costs and various risks. Therefore, it is crucial to minimize the deployment of agents that are either suboptimal or pose hazards. This is especially true in RL robotics control applications that are trained on massive amounts of simulated data.

Training on simulated data allows scaling in a virtual environment and leveraging large amounts of computing resources. However, this approach introduces a challenge: inconsistencies between the real world and the simulator. These discrepancies, termed the *sim-to-real* gap, can lead to learned policies overfitting to the simulator, resulting in potentially brittle controllers when ported to real-world hardware.

Detecting such overfitting to the simulator is not trivial and often necessitates real-world deployment, risking unsafe behavior in our deployed policy and prolonging the development cycle for new policies. In standard supervised learning, this issue is mitigated by the train-test split, where both training and test sets are sampled from the real-world distribution, enabling accurate performance evaluation on the test set. In contrast, RL generally involves sampling from an approximate distribution (the simulator), followed by evaluation on the same distribution, making it harder to detect overfitting concerning the underlying real-world distribution, as we never directly evaluate on it. As a result, a policy that performs well in the simulated environment may fail when deployed in the real world due to overfitting to the simulator's specific characteristics.

Consequently, evaluating a trained policy before deployment is a major challenge that we must address to enable the widespread use of RL algorithms in practice. Solely relying on the simulator where training occurs is insufficient; we need robust evaluation methods to detect potential overfitting and ensure adequate real-world performance.

Moreover, there are scenarios where hand-coded policies exist, but no simulator is available. These policies may still require tuning of hyperparameters, which can be costly and resource-intensive in the real world. While developing a simulator is one option for evaluation, it is expensive and inherently suffers from the sim-to-real gap. This further emphasizes the need for alternative evaluation techniques that do not depend on simulators. To sum up, we want to be able to evaluate the performance of our policies on the real-world distribution (or a very close distribution) without actually deploying them in the real world to save on resources.

Such an evaluation is possible with a family of techniques termed Off-Policy Evaluation (OPE). These techniques utilize a set of historical real-world data, often referred to as *behavior* or *off-policy* data, to estimate the underlying real-world distribution, thereby enabling the performance evaluation of a novel agent, also known as the *target* policy. Because this historical data is typically generated by different policies (the *behavior* policies) than those currently being evaluated, these techniques are termed off-policy techniques. Figure 1.1 illustrates the process of OPE.

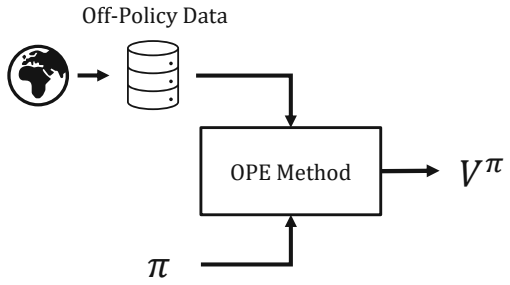


Figure 1.1: The process of OPE involves estimating the performance (V^π) of a policy π given historic off-policy data collected in the real world. Figure inspired by [FNN+21].

Crucially, we need to have access to historical datasets. These datasets can be collected by human experts or from previous policy prototypes. Alternatively, we may perform a small-scale real-world deployment of some of our agents and perform OPE only on the remaining agents and identify additional agents that would be worthwhile to be deployed.

Despite the advancements of OPE methods in recent years, their practical usability has been confined to simple classes of policies and simulated tasks. Recently, novel benchmarks have been proposed by [FNN+21], [QZG+22], and [VLA+21] to extend the validation of OPE methods on policies parameterized by deep neural networks. However, these benchmarks rely solely on simulated data and do not necessarily transfer to real-world robotics tasks. We believe validating OPE methods in real-world scenarios is

necessary to assess and reveal their effectiveness in robotics control. Only real-world data retains the complexity and uncertainty necessary to evaluate safe deployment and study the sim-to-real gap from the perspective of OPE. To this aim, we propose a novel application for autonomous racing with real-world F1TENTH miniature racecars.

This aim leads to the following research questions that are answered across this thesis:

1. How can OPE methods be effectively benchmarked in the F1TENTH environment?
2. Which OPE methods perform best in the F1TENTH environment?
3. How can OPE methods be enhanced to improve their practical application in the context of F1TENTH?
4. How effective is OPE in bridging the sim-to-real gap, when compared to simulation?

In this thesis, we answer these research questions by developing an OPE benchmark for the F1TENTH environment. We first collected real-world data with existing controllers in lab conditions. Then, we conduct an extensive experimental phase to compare the performance of various existing OPE methods. After identifying those approaches' shortcomings, we finally propose various adaptations tailored to our specific robotics applications, some of which can be applied more generally to other environments.

1.1 Outline

This thesis is structured in the following way:

Chapter 2 introduces the OPE problem formally and outlines the necessary background for discussing OPE. Specifically, the main approaches from literature concerning basic OPE techniques and previous benchmarks are introduced.

Chapter 3 discusses the specific F1TENTH robotics environment we investigate, including the definition of our agents, the process of off-policy dataset generation, and our evaluation dataset.

Chapter 4 presents the concrete OPE methods benchmarked in the real-world F1TENTH environment. Specifically, we discuss the adaptation and design choice to improve the existing approaches, including the novel TPDWIS estimator and a proof of its consistency.

Chapter 5 describes the experimental evaluation and reports our benchmark results. At the end of the chapter, we provide recommendations on which OPE method to employ.

Lastly, Chapter 6 summarizes our findings, discusses limitations, and provides direction for future work.

CHAPTER 2

Background and Related Work

This chapter introduces key concepts and methodologies essential to this thesis. It covers Markov Decision Processes (MDPs), the Off-Policy Evaluation (OPE) problem, relevant background on statistical estimators, various OPE methods, and examines prior benchmarks and their results concerning OPE methods.

2.1 Markov Decision Processes

A Markov Decision Process (MDP) is a mathematical framework for modeling stochastic decision-making processes. The framework is comprised of an environment and an agent that takes action and influences the state of the environment over time. Furthermore, the quality of decisions is modeled with a reward signal, a numerical value supplied to the agent at each timestep.

In this thesis, we consider finite-horizon MDPs that are characterized by a tuple (S, A, P, R, γ, T), defined as follows:

- S: Represents the state space, denoting all possible states the system can take. A specific state is denoted with s. Furthermore, an environment starts in an initial state s_1. As a concrete example, an approximate state of a racing car might be described according to its pose relative to the race track and the current velocities the car experiences. We generally also include the number of timesteps since the initial state in our state.

- A: The action space, encompasses all possible actions the agent can take to influence the next state of the system. A specific action is denoted with a. In our autonomous racing setting, the action space can be described by a steering command and a target velocity.

- P: The transition probabilities, $P : S \times A \times S \to [0, 1]$ denote the probability of reaching a subsequent state s_{t+1} from a state s_t by performing action a_t. Since the transitions are probabilistic, we can capture inherent uncertainty in our environment, such as state estimation errors and inherent stochasticity due to uncertain timings or unknown parameters.

- R: The deterministic reward function $R : S \times A \times S \to \mathbb{R}$ maps a transition, composed of state s_t, action a_t and next state s_{t+1} to a scalar value r_{t+1} , quantifying the immediate reward received upon state transition. For instance, with respect to racing agents, we might consider a progress reward that scales with the distance we cover towards our goal line between subsequent states. As we receive immediate feedback the reward can also be used to guide and train an agent to perform better in the future.

- γ: The discount factor $\gamma \in [0, 1]$ assigns more weight to rewards that occur sooner compared with delayed ones.

- T: Our termination function, $T : S \to \{0, 1\}$. Based on the state, this function marks whether the state is absorbing (1) or not (0). We call the transitions from the starting state to an absorbing state an *episode*. In a racing scenario, the termination function can be defined as colliding with the environment or getting close to it. As we deal with finite horizon MDPs every episode is also terminated after a set number of timesteps.

Figure 2.1 gives a visual overview of the interactions in a MDP.

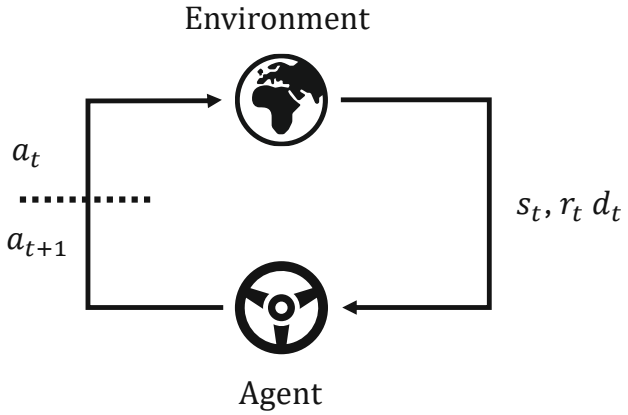


Figure 2.1: Depiction of an MDP. The environment provides the state s_t, reward r_t and termination signal d_t. Based on this, the agent emits an action a_{t+1}, which is again applied to the environment.

One fundamental assumption in MDPs is the **Markov property**, which states that the state at $t+1$ only depends on the state and action at t [SB18]. In our setting, we consider a stochastic policy $\pi : S \times A \to [0, 1]$ from which we sample an action at each timestep, $a \sim \pi(s)$. This setup provides a structured approach to model decision-making processes and their performance in terms of expected cumulative returns.

A *trajectory*, is a sequence of tuples $\tau = \{(s_1, a_1), \dots (s_i, a_i), \dots (s_T, a_T)\}$, generated by repeatedly sampling the next state from our transition probabilities $s_{t+1} \sim P(s_t, a_t)$, where the action is sampled from the stochastic policy, $a_t \sim \pi(s_t)$, as discussed above. We use s_t^τ and a_t^τ to denote the state and action within trajectory τ. If the identity of the trajectories is unimportant, we may omit the superscript in our notation. We denote a trajectory generated by a specific policy π as τ^π.

In line with this notation, we denote the reward at timestep t in the trajectory τ with r_t^τ. Given this notation, the return of a complete trajectory R^τ can be calculated as:

$$R^\tau = \sum_{t=1}^{T(\tau)} \gamma^{t-1} \cdot r_t^\tau \tag{2.1}$$

Here, we abuse notation slightly, as we supply our termination function $T(\cdot)$ with the trajectory (which includes actions), and it returns the timestep on which termination occurs, based solely on the state.

2.2 The Off-Policy Evaluation Problem

Given a fully specified MDP, we may compute the expected *performance* V^π of our *target policy* π, quantified by its expected cumulative reward. Formally, we can express this as:

$$V^\pi = \mathbb{E}[R^\tau | \tau \sim \pi] \tag{2.2}$$

However, the transition probabilities P, the reward function R or at least the termination function T are unknown or very hard and costly to infer in most interesting settings. Hence, we cannot sample trajectories and compute R^τ directly. Off-Policy Evaluation (OPE) is a collection of techniques to estimate V^π leveraging pre-existing data.

In particular, we assume access to an off-policy dataset, consisting of trajectories generated with various *behavior policies* $\{\pi_1, \dots \pi_i, \dots \pi_M\}$. The trajectories may consist of completely different behavior policies, previously deployed solutions, or human input in the case of teleoperation. We formally denote a dataset comprised of trajectories of such behavioral policies as:

$$\mathcal{D}_b = \{\tau_1^{\pi_1}, \dots, \tau_{N_1}^{\pi_1}, \dots, \tau_1^{\pi_i}, \dots, \tau_{N_M}^{\pi_M}\} \tag{2.3}$$

According to this notation, $\tau_j^{\pi_i}$ denotes the j^{th} trajectory collected under policy π_i.

As implied by the off-policy paradigm, our dataset $\mathcal{D}_b$ generally does not contain trajectories from the target policy π. Several OPE methods have been developed to estimate the target performance by leveraging the behavior data and considering the difference with the target policy. In Section 2.4, we will present the most relevant works for our setting.

2.3 Statistical Estimators

Solving the OPE problem requires estimating accurate statistics from off-policy data. Therefore, statistical estimators are at the core of OPE methods. These offer different accuracy in the estimation, trading off bias and variance. This section introduces fundamental concepts of statistical estimators relevant to the discussion of OPE methods.

The choice of OPE technique involves navigating the trade-off between bias and variance - a fundamental concept in statistical learning theory [GBD92, JWH+13]. In this context, bias refers to the error introduced by the assumptions made by the model to simplify the real-world problem. Variance, on the other hand, refers to the error that occurs due to the model's sensitivity to the specific samples used to generate the training dataset. An estimator with a low variance is more likely to return an acceptable result with less training data. Accepting some bias, especially when considering smaller datasets, can reduce variance - potentially reducing overall error [SB18, JWH+13].

However, many estimators discussed throughout this thesis are also *consistent*. Meaning that the bias approaches zero as our training dataset grows to infinity. These concepts are formally defined in the following, as in [Tho15].

Definition 1 (Almost Sure Convergence)**.** *A sequence of random variables $(V_n)_{n=1}^{\infty}$ is said to converge almost surely to a random variable V if:*

$$\Pr\left(\lim_{n\to\infty} V_n = V\right) = 1.$$

Here, Pr *denotes the probability measure, quantifying the likelihood of a particular event occurring.*

Definition 2 (Unbiasedness)**.** *Let V^π be a real-valued parameter of interest and let V be a random variable. V is said to be an unbiased estimator of V^π if and only if*

$$\mathbb{E}[V] = V^\pi.$$

Definition 3 (Consistency)**.** *Let V^π be a real-valued parameter of interest and let $(V_n)_{n=1}^{\infty}$ be an infinite sequence of random variables. V is said to be a (strongly) consistent estimator if and only if*

$$(V_n)_{n=1}^{\infty} \xrightarrow{a.s.} V^\pi.$$

Where $\xrightarrow{a.s.}$, denotes almost sure convergence, as in Definition 1.

Intuitively, an unbiased estimator will match the true value in expectation, while the error of a consistent estimator will approach zero as more and more samples are made available. In our setting, the random variable V is our specific estimator, which is stochastic over the trajectories in our dataset $\mathcal{D}_b$. A sequence of estimators is defined by adding additional trajectories to our dataset $\mathcal{D}_b$.

Importantly, being unbiased does not guarantee convergence as we draw more and more samples from V. To make this clear, let us consider an example of an unbiased but inconsistent estimator inspired by [Sta12].

Given an on-policy dataset $\mathcal{D}$ consisting of trajectories generated with π, we can choose a return R^τ of one specific trajectory, τ_i^π, as our estimator for V^π. In that case, clearly, $\mathbb{E}[R^\tau] = V^\pi$. However, as more samples are added to $\mathcal{D}_b$, the error $R^\tau - V^\pi$ will remain constant, as our estimator does not depend on the rest of the dataset. To conclude, the estimator is unbiased - but not consistent. For examples of other combinations of (un)biasedness and (in)consistency, the reader is referred to [Tho15].

2.4 Off-policy Evaluation Methods

This section discusses various methods to address the OPE problem presented in Section 2.2. To this aim we structure this section into four parts, with each of them addressing one family of OPE methods:

- **Model-based (MB)**: these techniques directly approximate the transition probabilities $\mathcal{F} \approx P$ and estimates the performance of the target policy π by performing Monte-Carlo trajectory rollouts on $\mathcal{F}$.
- **Fitted-Q-Evaluation (FQE)**: this family of model-free OPE methods learns a Q-function to predict the target performance from state/action pairs by ensuring that transitions adhere to the Bellman equation.
- **Importance Sampling (IS)**: a family of classical statistical estimators that reweights the behavior performances based on the likelihood of the observed actions under the target policy compared to the behavior policy.
- **Hybrid**: These methods combine IS methods with one of the other estimators.

2.4.1 Model-based

Model-based (MB) methods rely on an approximation $\mathcal{F}$ of our true transition probabilities P. This approximate model can be derived from expert knowledge or learned from data, as we will explain later in this section. Having an approximation $\mathcal{F}$, we can query it to predict the next state:

$$s_{t+1} \sim \mathcal{F}(s_t, a_t) \tag{2.4}$$

We can now address the counter-factual question of estimating the performance π by performing Monte-Carlo trajectory rollouts. Therefore, our estimate is given as:

$$V_{mb}^{\pi} = \frac{1}{|\mathcal{D}_b|} \sum_{s_1^\tau \in \mathcal{D}_b} \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=1}^{T(\tau)} \gamma^{t-1} R(s_t^\tau, a_t^\tau, s_{t+1}^\tau) \middle| s_{t+1}^\tau \sim \mathcal{F}(s_t^\tau, a_t^\tau), a_t^\tau \sim \pi(s_t^\tau) \right] \tag{2.5}$$

where we start each rollout at the starting points of the trajectories contained in $\mathcal{D}_b$. Hence, our estimator returns the value estimate conditioned on the provided starting points. Since both our transition function $\mathcal{F}$ and our policy π may be stochastic, we take the average over N Monte-Carlo rollouts per starting point, such that we can estimate the true expected value. This formulation also relies on of both the reward and termination functions, R and T, respectively, being known.

$\mathcal{F}$ can be inferred from pre-recorded datasets. Here, we can make one larger distinction - human-derived dynamics equations, typically in the form of differential equations, and dynamics learned with function approximation methods directly from the recorded dataset. The first kind, typically known as simulation, is generally not inferred from our concrete dataset $\mathcal{D}_b$. For the second kind, we explicitly learn a transition model from the data in $\mathcal{D}_b$.

Simulation

Although simulation is often not mentioned alongside traditional OPE methods in the literature, it fits into this category. The reason for this is straightforward: if we have access to dynamics equations that model our underlying system, these are generally built on vast amounts of empirical data. This apparent oversight in previous literature may stem from previous works treating the simulator as the primary environment under study, contrasting with the work in this thesis, wherein the agent is deployed in the real world.

While simulation can be a very effective technique, and many simulators are available for various domains like robotics [BB20, WWG+21, CLW+07], they require extensive domain knowledge and resources to develop. In our specific setting of autonomous racing, a dedicated simulator is available [BB20]. However, achieving the most accurate results requires precise tuning of over 15 parameters, some of which are not readily inferable. Despite the setting of these parameters, a residual sim-to-real gap persists. This gap is primarily due to the simulator's reliance on an approximate representation of the real world, such as using a single-track dynamics model, which cannot capture every nuance of real-world conditions.

As already touched upon in Section 1, this issue is commonly known as the sim-to-real gap [SFMP21, ZQW20]. Many techniques have been proposed to address this gap (see [SFMP21] and [ZQW20] for comprehensive reviews), with varying degrees of dependence on real-world deployments. In any case, the performance of simulation directly depends on the quality of our human-derived dynamics equations and the used parameters.

Learning $\mathcal{F}$

Learning the transition probabilities has been extensively studied under the umbrella of Model-based RL and has yielded promising results, especially when parameterizing $\mathcal{F}$ as a neural network and training it in a probabilistic fashion with log-likelihood loss [MBJ17, ZPN+21, CCML18, FNN+21, US21]. Some techniques that have, to our knowledge, not yet been explicitly explored in the OPE setting but are readily transferable can be found in the survey by [MBP+23]. For instance, multiple works (e.g., [AN04] and [ACML18]) have found that the state prediction accuracy can be increased by taking longer parts of trajectories into account, either during training [AN04], or predicting multiple time-steps at once [ACML18]. In principle, any already existing method for learning dynamics functions from an offline dataset is applicable to OPE.

If the reward R and termination T functions are not known beforehand, we might additionally learn them from data [ZPN+21, KN20]. For instance, we can learn the termination and reward functions as additional models [KN20]. Other work shows that learning them as an additional state dimension can be beneficial [ZPN+21, MBP+23]. However, in our setting, we assume both the reward and termination functions to be known, and only focus on learning the transition dynamics.

2.4.2 Fitted-Q-Evaluation

Fitted-Q-Evaluation (FQE) was introduced by [LVY19] specifically for the OPE setting. FQE predicts V^π directly with a Q-function, parameterized as a neural network. Algorithm 2.1 shows the training procedure for FQE. Training relies on the Bellman backup, as seen in line 4 of Algorithm 2.1.

The trained FQE method takes in a state-action tuple and directly predicts the expected return for it. Hence for our dataset, the FQE-estimator can be written as:

$$V^\pi_{FQE} = \frac{1}{|\mathcal{D}_b|} \sum_{\tau \in \mathcal{D}_b} \frac{1}{N} \sum_{i=1}^{N} [Q(s^\tau_1, a_1) | a_1 \sim \pi(s^\tau_1)] \tag{2.6}$$

Since our policy π is stochastic, we can sample multiple actions and compute Q estimates for each, averaging to improve the estimator. N is the number of samples we draw from our stochastic policy.

2.4.3 Importance Sampling

Importance Sampling (IS) is a longstanding method in statistics that has been successfully transferred to the OPE problem [PSS00, Tho15]. We will start by discussing IS in general, then its application to MDPs, and last presenting different IS methods.

General formulation. In the IS framework, we are interested in estimating the expected value of a function f over samples X, distributed according to some target

Algorithm 2.1: Simplified FQE training, code adapted from [LVY19].

Input: Training Dataset $\mathcal{D}$, Actor π, number training steps T, discount factor γ
Output: Trained FQE Model θ

θ = initialize(θ)
for 1 **to** T **do**
  $\{s, a, r, s', d\} \sim \mathcal{D}_b$ `// Sample state, action, single-step reward, next state and boolean done`
  $y \leftarrow r + (1 - d) \cdot (\gamma Q(s', \pi(s')))$ `// compute targets`
  $\hat{y} \leftarrow Q(s, a)$ `// compute Q prediction`
  $loss \leftarrow loss\_fn(y, \hat{y})$ `// compute some divergence measure`
  $\theta \leftarrow \theta - \alpha \nabla_\theta loss$ `// update network`
end
return θ

distribution p_t:

$$\mathbb{E}[f(X)|X \sim p_t]$$

having access to samples from a behavior distribution p_b, for which we can therefore compute $\mathbb{E}[f(X)|X \sim p_b]$. The IS estimator allows us to adjust our computations from p_b to effectively estimate the desired expectation under p_t. The IS estimator V'_{IS} is given as [Tho15]:

$$V'_{IS} = \frac{1}{|X|} \sum_{x \sim p_b}^{|X|} \frac{p_t(x)}{p_b(x)} f(x) \tag{2.7}$$

with $|X|$ being the total number of samples x we draw from p_b.

We call $\frac{p_t(X)}{p_b(X)}$ the *Importance Sampling ratio*. This ratio reweights the returns of f, having a weight larger than one for values that are more likely under our target distribution and a weight less than one for values that are less likely. If the likelihood is the same for each distribution, the ratio will be one - hence, no reweighting occurs for such samples. Figure 2.2 shows a visualization of how the density of our function f is adjusted from p_b to p_t.

IS is an unbiased estimator under the following assumption on the probability distributions:

Assumption 1 (Absolute continuity)**.** $p_t(x) \neq 0 \implies p_b(x) \neq 0$, $x \in X$.

To derive the IS estimator and show its unbiasedness, we can follow the derivation of [Tho15]:

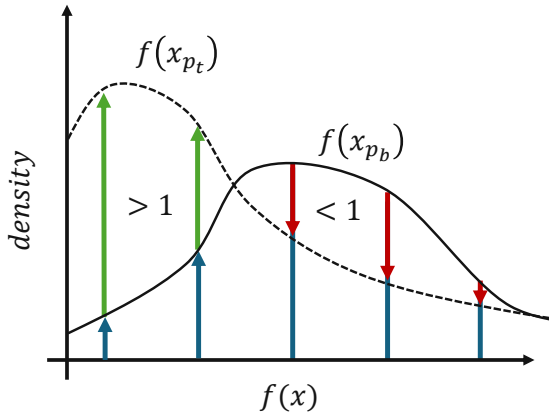


Figure 2.2: The figure presents the initial distribution of f, represented by a solid line, derived from samples taken from p_b, denoted as $f(x_{p_b})$. This distribution is modified using the IS ratio to align with the target distribution $f(x_{p_t})$, illustrated by a dashed line. Regions labeled with > 1 and < 1 indicate areas where the IS ratio exceeds or falls below one, respectively.

$$\begin{aligned}
\mathbb{E}[f(X)|X \sim p_t] &= \sum_{x \in supp\ p_t} p_t(x) f(x) \\
&\overset{(1)}{=} \sum_{x \in supp\ p_b} p_t(x) \frac{p_b(x)}{p_b(x)} f(x) \\
&= \sum_{x \in supp\ p_b} p_b(x) \frac{p_t(x)}{p_b(x)} f(x) \\
&= \mathbb{E}\left[\frac{p_t(X)}{p_b(X)} f(X)|X \sim p_b\right]
\end{aligned} \tag{2.8}$$

In step (1) we multiply with $1 = \frac{p_b(x)}{p_b(x)}$. This manipulation requires Assumption 1; otherwise, the division by p_b would be undefined in cases where $p_b = 0$, making it impossible to compute the IS estimator for such data points. Furthermore, in addition to being unbiased, as shown above, IS is also consistent [Tho15].

Importance Sampling for Off-Policy Evaluation. We will apply IS in our MDP context, as for example done by [Tho15], [SB18], and [PSS00]. Our data consists of trajectories from our behavior policies, $\tau \sim \pi_i$, with f being replaced by our reward function. Since our rewards depend on the state history, we are interested in the IS ratio reflecting the relative probability of a trajectory under the target policy, given the behavior policy. Therefore, the IS ratio for a single timestep should reflect the likelihood of both agents ending up in the same state s_{t+1}. Hence, the single-step IS ratio is given

as [Tho15, SB18]:

$$p_t^\tau = \frac{\pi(a_t|s_t)P(s_{t+1}|s_t,a_t)}{\pi_b(a_t|s_t)P(s_{t+1}|s_t,a_t)} = \frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)} \tag{2.9}$$

Here, the transition probabilities, $P(s_{t+1}|s_t,a_t)$ cancel out, simplifying our expression to the probability of picking an action under our target distribution divided by the probability under our behavior distribution. We can now restate Assumption 1 for our specific MDP case:

Assumption 2 (absolute continuity MDP)**.** $\pi(a_t|s_t) \neq 0 \implies \pi_b(a_t|s_t) \neq 0 \quad s_t \in S, a_t \in A$.

Assumption 2 has a serious consequence on the agents that we can investigate in this thesis if we want to employ IS in a straightforward manner: our agents need to be stochastic over the action space. Otherwise, we cannot satisfy the assumption. Consider a deterministic agent π that takes a different action in s_t than the deterministic agent π_b. In that case $\pi(a_t|s_t) \neq 0$, but $\pi_b(a_t|s_t) = 0$, making Equation 2.9 undefined.

When considering the cumulative IS ratio of trajectories till timestep t, we write:

$$p_{1:t}^\tau = \prod_{t'=1}^{t} p_{t'}^\tau \tag{2.10}$$

If we are interested in the IS ratio of a complete trajectory, we denote it with $p_{1:T(\tau)}^\tau$, following the notation for $T(\cdot)$ returning the length of a trajectory, as established earlier.

Variants of Importance Sampling

Ordinary Importance Sampling Ordinary Importance Sampling reweights the returns of all trajectories in $\mathcal{D}_b$. The reweighting is performed according to the cumulative product of the trajectory-wise IS ratios as in Equation 2.10. Our ordinary importance sampling estimator V_{IS}^π is therefore given as:

$$V_{IS}^\pi = \frac{1}{|\mathcal{D}_b|} \sum_{\forall \tau \in \mathcal{D}_b} R^\tau \cdot p_{1:T(\tau)}^\tau \tag{2.11}$$

As for example shown by [Tho15], V_{IS}^π is an unbiased and consistent estimator. However, the estimator can have very high variance [PSS00, Tho15, SB18, LKTF20].

Per-Decision Importance Sampling

The high variance associated with ordinary IS can sometimes be reduced by using Per-Decision Importance Sampling (PDIS), as introduced by [PSS00].

In the per-decision approach, rather than applying the cumulative IS ratio over an entire trajectory, each reward at timestep t is adjusted according to the cumulative IS ratio up to that specific step. The estimator V^{π}_{PDIS} is defined as:

$$V^{\pi}_{PDIS} = \frac{1}{|\mathcal{D}_b|} \sum_{\forall \tau \in \mathcal{D}_b} \sum_{t=1}^{T(\tau)} \gamma^{t-1} \cdot r^{\tau}_{t} \cdot p_{\tau,1:t} \tag{2.12}$$

This variant, also known as step-wise IS, does not compromise the estimator's unbiasedness or consistency [PSS00, Tho15], but offers reduced variance in some cases.

Weighted Importance Sampling

The range of estimates from the IS and even the PDIS estimators can still be very high due to the IS ratios having an extensive range of possible values [PSS00, Tho15, SB18]. In many settings, these estimators will return values far below (like in our setting, see Section 4.3) or above any return observed in the dataset.

This large reward range can be remedied by utilizing Weighted Importance Sampling (WIS). This is achieved by normalizing the IS ratios, such that they sum up to one [Tho15, PSS00, SB18]. The WIS sampling estimator is given by:

$$V^{\pi}_{WIS} = \sum_{\forall \tau \in \mathcal{D}_b} R^{\tau} \frac{p^{\tau}_{1:T(\tau)}}{w} \tag{2.13}$$

$$w = \sum_{\forall \tau \in \mathcal{D}_b} p^{\tau}_{1:T(\tau)} \tag{2.14}$$

Henceforth, we refer to the fractional part of Equation 2.13 as the normalized Importance Sampling ratio. With the normalized IS ratios now adding up to one, the estimated value of V_{WIS} falls within the range of the highest and lowest cumulative rewards seen in the associated training data.

Although WIS is no longer unbiased, it is still consistent, with the bias converging asymptotically to zero under Assumptions 2 [Tho15, SB18]. Additionally, WIS generally decreases variance significantly, compared to the previous IS estimators. This is very important in cases where there is not a lot of data available [Tho15].

2.4.4 Hybrid-methods

Hybrid methods combine IS with another method. The other method is generally a neural network based approach that allows us to estimate $V^{\pi}(s)$ and $Q^{\pi}(s,a)$ [JL16, FNN+21, VLA+21]. Hence, both MB and FQE-based methods are applicable in this context.

Generally, IS is utilized with a dataset where the per-step rewards have been modified in some way by the non-IS-based method. For instance, the Doubly Robust Estimator (DR) estimator adjusts the per-step rewards with the time-difference error according to $V^{\pi}(s)$ and $Q^{\pi}(s,a)$. More details and the concrete formalism for DR are given in Section 4.4. Other advanced hybrid estimators, like MAGIC [TB16], improve results further by optimizing for the horizon until which IS is applied in a trajectory, after which only the value function is utilized [TB16].

This thesis explores the performance of various versions of DR, utilizing FQE with different IS estimators.

2.5 Off-Policy Evaluation Benchmarks

The performance of different OPE methods is inherently tied to the specific task and the off-policy data. In order to advance the state of the art in OPE and enable fair comparisons multiple benchmarks have emerged, which are discussed in this section.

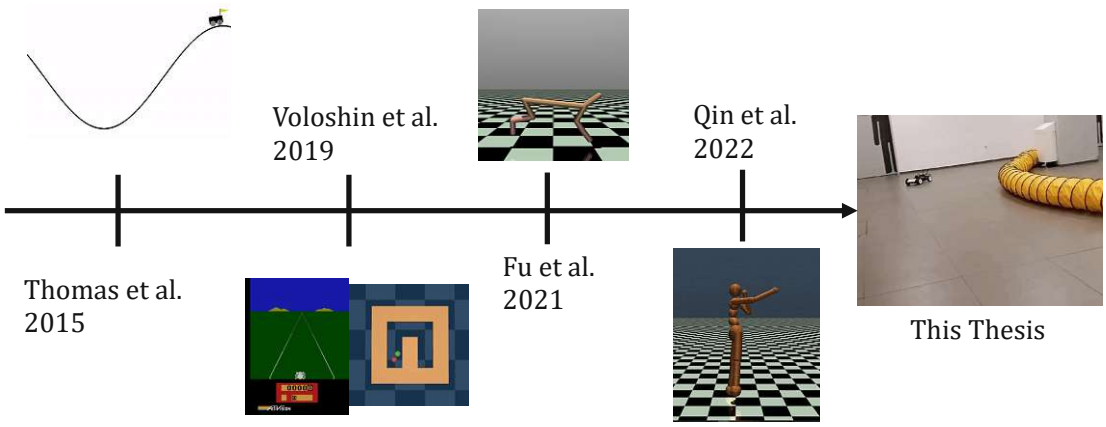


Figure 2.3: Example environments from previous OPE benchmarks. This depiction aims to show the approximate evolution of OPE benchmarks, culminating in the real-world OPE benchmark introduced in this thesis. However, it needs to be pointed out that there is some overlap in the tasks of previous benchmarks, and OPE techniques might have already been applied to the shown environments earlier than depicted.

Caltech OPE Benchmarking Suite (COBS): [VLA+21] benchmark OPE methods in a total of 8 different simulation environments, ranging from a simple so-called 'graph' environment with a single integer state and binary actions to more complex and challenging ones, like Mountain Car and Atari (Enduro). In their work, they benchmark a total of 33 different OPE methods in these environments; most of the methods are different combinations of MAGIC and DR with function approximation estimators. They find that the performance of OPE methods, perhaps unsurprisingly, depends heavily on

the environment and the dataset-target policy mismatch, with no method consistently outperforming the others across experiment settings [VLA+21]. Regarding IS methods, they consider IS, PDIS, WIS, and PDWIS, with their experiments indicating PDWIS performing best amongst them [VLA+21]. They also find that FQE can perform well and is often amongst the best methods, especially in limited data domains. Regarding hybrid methods, MAGIC with FQE performs best, generally outperforming its DR counterpart [VLA+21]. Last, in their experiments, where they consider a naive Model-based method, this method performs worst among the considered methods since small errors in the transition model tend to accumulate [VLA+21]. As evaluation metrics, COBS considers Near-Top Frequency, measuring how often a method tends to be in the top 10% of the best performing methods, as measured with relative mean squared error across multiple seeds [VLA+21].

Near real-world Benchmark for offline RL (NeoRL): [QZG+22] proposes a complete pipeline for training and evaluating offline RL algorithms. Importantly, they emphasize the need for realistic, real-world-like data generation. NeoRL contains a total of 52 tasks across 7 domains [QZG+22]. All of the tasks are simulation-based. Furthermore, since NeoRL focuses on providing a whole pipeline for offline RL in general, they only evaluate two OPE methods, namely FQE and WIS. Both methods severely underestimate the true return across all tasks, ranking the seven investigated policies incorrectly, with both preferring the third best-performing policy. Furthermore, they note that WIS and FQE often cannot distinguish bad-performing policies from good ones [QZG+22].

Deep OPE Benchmark (DOPE): [FNN+21] propose the Deep OPE Benchmark (DOPE) focusing on complex continuous robotic domains, incorporating established benchmarks in this domain, namely RL Unplugged [GWN+20] and D4RL [FKN+20]. Similar to the previously discussed benchmarks, they only consider simulation data. They benchmark FQE-L2, DR, a version of PDWIS, VPM, and DICE on all their tasks and additionally evaluate an Autoregressive Model-based method and FQE-DD on a subset of tasks [FKN+20]. VPM and DICE do not apply to the setting investigated in this thesis since they aim to estimate the action log probabilities, which we assume are known (see Chapter 3.1.2 for more information on the setting we investigate). Their results indicate that the versions of FQE and the MB methods perform best, generally outperforming DR slightly. However, they assume the action probabilities are unavailable. This most likely explains the under-performance of methods that rely on IS. Across all benchmarks FQE, DR, and as indicated by DOPE, MB methods can perform well in OPE, depending on the environment and setting.

Figure 2.3 shows a selection of tasks considered in the previously discussed OPE benchmarks and the task considered in this thesis. Importantly, **none** of the benchmarks known to the authors consider real-world robotics data or investigate the potential for OPE to bridge the sim-to-real gap in real-world robotics.

CHAPTER 3

Experimental Setup

This chapter details the creation and the properties of the F1TENTH offline dataset studied throughout this thesis. First, Section 3.1, describes the setup of our mobile robot actor and the associated Markov Decision Process of our environment. Section 3.2 describes the design and parametrization of the agents deployed on the robot platform. Section 3.3 discusses the collection of real-world data. Last, Section 3.4 discusses properties of the resulting dataset, including the number of agents contained, the trajectories and the split into train and evaluation datasets.

3.1 F1TENTH Platform

In order to study the performance of OPE methods in real-world robotics applications, we adopted the F1TENTH robotics platform, introduced by [OZKM20]. The platform is ideal for testing mobile robotics applications due to its simple accessibility and relatively low-cost hardware. The simulator, integrated in a training environment `f1tenth-gym`, supports the Robot Operating System (ROS) [QCG+09] and implements the gymnasium API [BCP+16]. The accessible features of this platform have made it an attractive choice for deploying and studying Reinforcement Learning (RL) methods in the real world, with various research projects having already explored its potentials [BBB+22, EEJ21a, EEJ21b]. Furthermore, the F1TENTH platform is regularly used in racing competitions, attracting a diverse range of research activities [BZL+22].

3.1.1 F1TENTH Actor

In this section, we describe the hardware of the platform and the auxiliary software.

Hardware: We conducted our experiments on *car-2* of the TU Wien ARC racing fleet. This vehicle is built according to the official F1TENTH guidelines [F1T23].

For state inference and navigation, we employ a the Lidar Hokuyo 10LX 2D [Hok15]. The LIDAR has a 270-degree field of view (FOV) with a total of 1081 LIDAR beams spread evenly across it. LIDAR data is made available in a best-effort manner with an average update rate between 30-40Hz. As IMU sensor, we employ a Sparkfun Razor-9DOF IMU [Spa19]. The IMU data is published with an average of 100Hz. As computing platform, we adopted a Nvidia Jetson Xavier NX, running Xubuntu. Figure 3.1 shows a vehicle equivalent to the one utilized for this thesis.

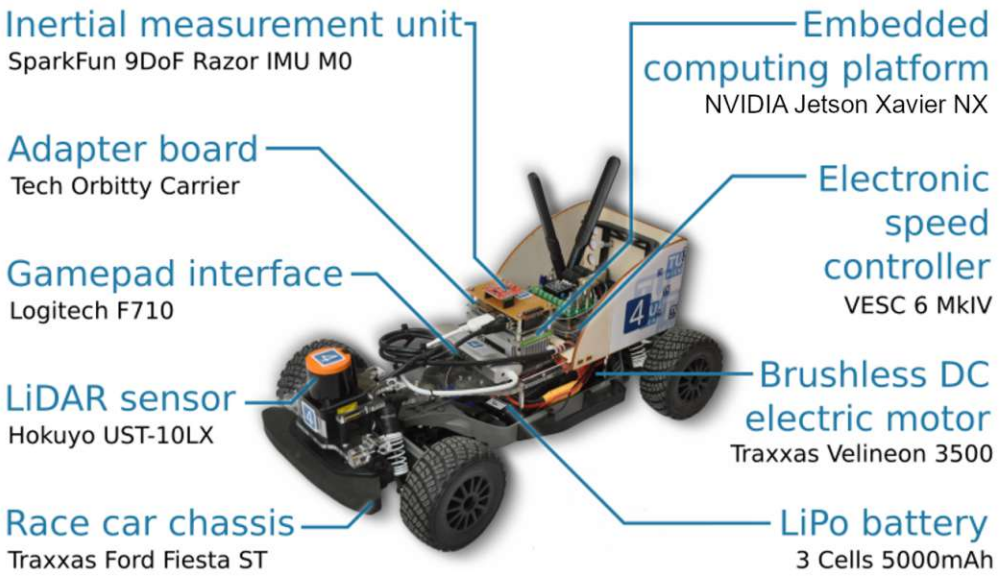


Figure 3.1: A F1TENTH racing vehicle, equivalent to the one employed in this thesis. Image minimally adapted from [GB24].

Auxiliary Software: The vehicle software uses ROS2 Foxy to exchange data between sensors and actuators. Figure 3.2 provides an overview of our ROS2 setup for deploying agents in the real world. Our implementation of the F1TENTH racing stack serves as the primary abstraction layer for sensor and actuators interaction. This stack encapsulates all necessary ROS2 nodes to deploy the robot. Amongst the most relevant nodes, we have the Particle Filter, which is responsible for pose estimation from the raw LIDAR data; the IMU node handling the calibration and forwarding of the IMU data; and the Teleop Node for manual teleoperation and interventions.

The second major component shown in in Figure 3.2, the Agent Wrapper, has been developed specifically for this thesis. The wrapper's primary purpose is to provide the same interface in the real world as in simulation, such that executing the same agent in the real world and simulation is possible. To this aim, the Observation Dict Assembler pre-processes the received real-world observations and provides them in a unified dictionary format to the experiment controller and, subsequently, the agent. The Agent Wrapper is also responsible for data collection and controlling trajectory rollouts

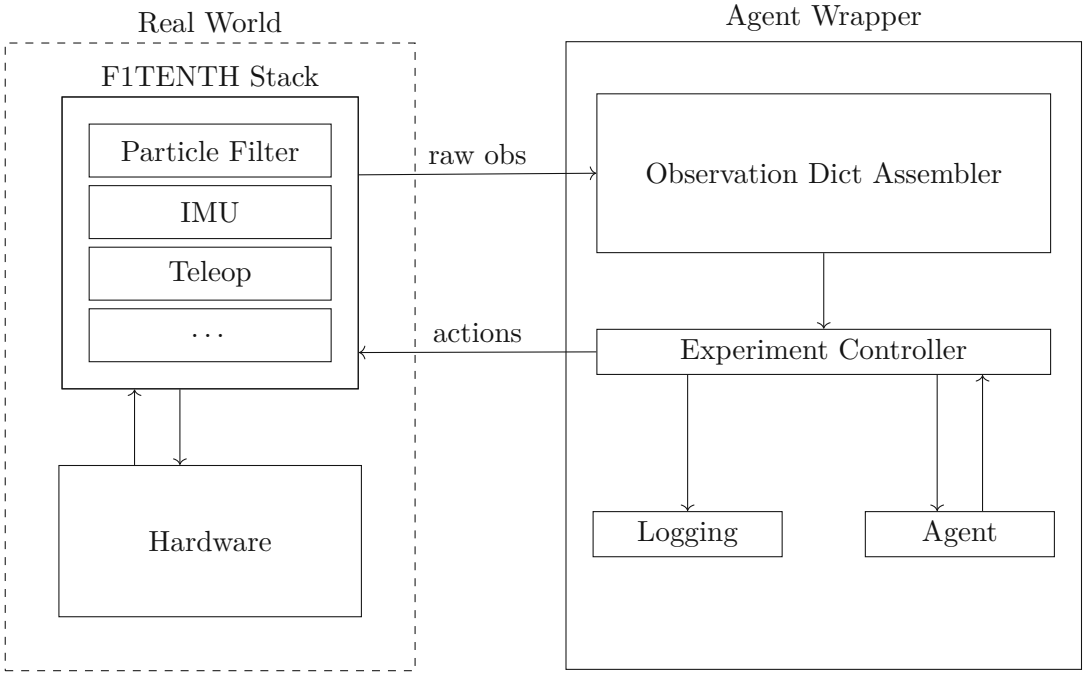


Figure 3.2: ROS2 architecture for real-world data-collection. The left part of the diagram, in the 'Real World' category, shows the F1TENTH software stack to run the vehicle. The Agent Wrapper, on the right side, implements a unified interface for our agent and controls real-world dataset collection.

as described in more detail in Section 3.3. A modified version of the Agent Wrapper exists for simulation, which takes in the simulation observations and provides the same observation to the agent as the real-world Agent Wrapper.

3.1.2 F1TENTH Offline Dataset Environment

In addition to an actor on which we can deploy an agent, our environment is defined by the racing track on which the vehicle drives. This thesis focuses on a single real-world racing track. The track, as obtained with LIDAR-based SLAM, specifically with the ROS2-slam toolbox [MJ21], is shown in Figure 3.3.

In the following, we describe the concrete F1TENTH MDP, including action space, state space, state-space transitions, and the investigated reward functions.

Action Space

The previously discussed F1TENTH stack takes a target *steering angle* in radians (rad) and a target *velocity* in meters per second (m/s) as an input. The agent wrapper provides a target velocity between 0.5 and 5.0 m/s and a steering angle. Subsequently, the racing stack attempts to match these targets. However, the agents we employ only provide

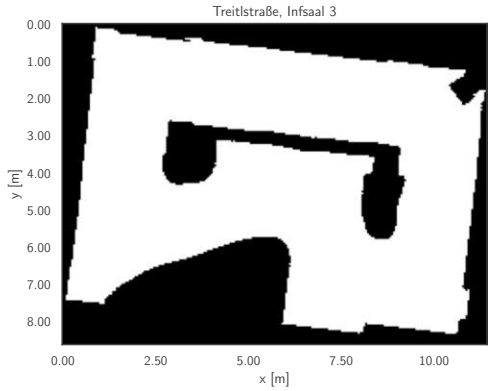


Figure 3.3: Our map as inferred by LIDAR-based SLAM. The driving direction is always clockwise. White areas indicate areas where driving is possible. Black areas are outside of the x/y state space.

changes in action, or *delta actions*, capped at ±0.3 for both steering and velocity. Hence, our agent wrapper calculates the absolute target commands from the previous actions.

Observation Space

Our agents are provided with an observation dictionary that contains the data shown in Table 3.1. To keep our agents stateless, as required by the OPE techniques employed, we include the previous absolute actions in the observations.

Observation	Unit	Inference	range
LIDAR Ranges	m (normalized)	LIDAR Sensor	[0,1]
position x	m	computed	$[-\infty, \infty]$
position y	m	computed	$[-\infty, \infty]$
position θ	rad	computed	[0,6.283]
linear velocity x	m/s	IMU Sensor	$[-\infty, \infty]$
linear velocity y	m/s	IMU Sensor	$[-\infty, \infty]$
angular velocity z	m/s	IMU Sensor	$[-\infty, \infty]$
previous steering	rad	stored	[-0.418, 0.418]
previous velocity	m/s	stored	[0.5,5.0]
timestep	-	computed	[0,250]

Table 3.1: The observation dictionary provided to each agent.

- **LIDAR Ranges:** To reduce the dimensionality, we down-sample the original 1081 rays to 54. Next, we pre-process them by cutting off ranges longer than 10 meters and normalizing the ranges to fall in the intervals of 0 and 1.

- **Position:** The position is defined with respect to our map origin. For pose estimation, we utilize a particle filter [WK18]. The particle filter computes the pose by ray-casting from each particle in a particle cloud, generated randomly around an estimated pose, picking the particle that has the highest similarity with our LIDAR rays. We use the default parameterization of the particle filter as used in previous racing events.

- **Velocities:** The velocity vector is estimated with the IMU.

- **Previous Absolute Actions:** Since our agents have to be stateless and we are employing delta-actions, we have to provide our agents with the previous absolute steering and velocity targets. These are initialized in the first timestep to a neutral steering of 0.0 and an initial velocity of 0.5 m/s in all trajectory rollouts we perform. After this initial timestep, they are always equivalent to the action supplied to the racing stack in the previous timestep.

- **Timestep** The timestep in the MDP is provided, as it is required for accurately modeling episodic MDPs.

Due to the asynchronous nature of ROS applications, the observation dictionary for real-world inference is updated in a best-effort manner. This means that as soon as new data is registered, we update the corresponding entry in the observation dictionary. In the real-world case, the observations provided to the agent are subject to noise and delay. Especially at high speeds, the delay becomes a significant issue for the agent, acting on old information. On the other hand, data provided by our utilized `f1tenth-gym` simulator [OZKM20] is largely noise-free (some small noise is applied to the LIDAR rays), and state estimation is exact. We discuss this sim-to-real gap and resulting performance differences in greater detail in Section 5.2.1.

State-space Transitions

Operating in a continuous real-world environment, we discretize time to model transitions with a specific granularity, choosing a frequency of 20Hz for both logging data and invoking our agent. Importantly, given the real-world nature, action execution is delayed, and observations will also be old. While we log the timestamps of both the action emission and state estimation and make it easy to access them in our dataset environment, we disregard them in the rest of this thesis, leaving a potential area for future work. Figure 3.4 shows the difference between the timestamp of state estimation and action emission for one trajectory. Note that the actual time delta is significantly higher since there are also delays associated with data capture at the sensor and forwarding the required data to our state estimator. As our system has no real-time guarantees, we occasionally experience very high delays, as visible in Figure 3.4.

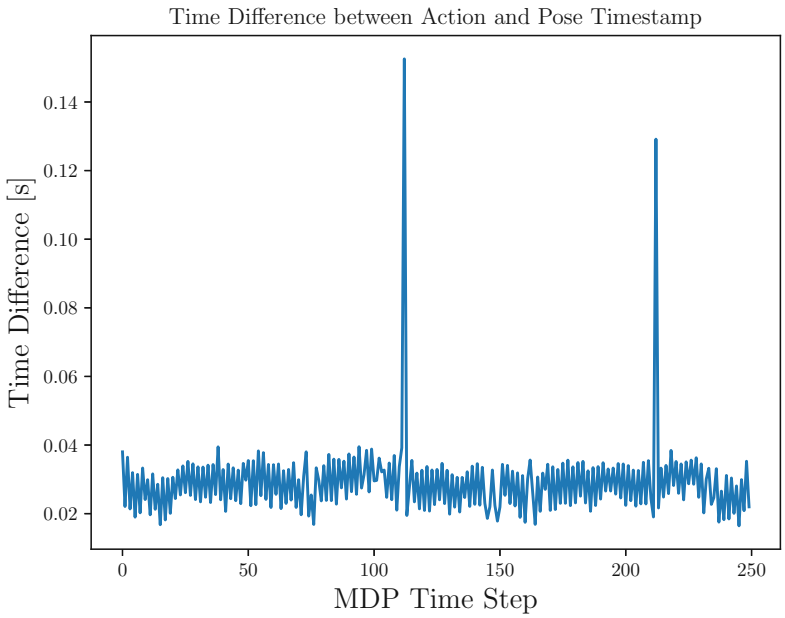


Figure 3.4: Delta-time between state estimation and action emission in seconds vs. timestep for one sample trajectory rollout. The spikes visible in the figure are large time delays between state estimation and action emission. As we are on a non-real-time computing platform these might occure due to factors such as high congestion on the system.

Reward Functions

To complete the definition of our MDP, we discuss the reward functions. We investigate four different reward functions to study the reward's influence on our OPE estimate. In cases of early termination, the reward for the remaining timesteps is set to zero. For better efficiency, the rewards are computed offline by processing each state of the collected trajectory.

- **Progress Reward:** The Progress Reward R_t^{Progress} is assigned to an agent based on the distance traversed between two consecutive time steps t and $t+1$ along a predefined centerline. Formally, let progress(s) be a function that returns the progress along the centerline given a state s. The Progress Reward is defined as:

$$R_t^{\text{Progress}}(s_t, a_t, s_{t+1}) = (\text{progress}(s_{t+1}) - \text{progress}(s_t)) \cdot C_p \tag{3.1}$$

  where C_p is a constant that scales the reward. We set $C_p = 100$ in our experiments. The larger the distance an agent traverses between t and $t+1$ with respect to a predefined centerline, the higher the reward.

- **Checkpoint Reward:** The Checkpoint Reward $R_t^{\text{Checkpoint}}$ is a fixed reward C_c assigned to an agent upon reaching predefined static checkpoints along the centerline of a track. Specifically, let there be N checkpoints, indexed as $\{1, 2, \ldots, N\}$, positioned along the centerline of the track. For each checkpoint $i \in \{1, 2, \ldots, N\}$, the agent receives the reward C_c the first time it reaches the checkpoint during a lap. Formally, the Checkpoint Reward is defined as:

$$R_t^{\text{Checkpoint}}(s_t, a_t, s_{t+1}) = \begin{cases} C_c & \text{if progress}(s_t) \text{ reaches checkpoint } i \text{ for the first time} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

  For the Checkpoint Reward, we define 10 static checkpoints along the centerline of the track. Upon reaching a checkpoint for the first time during a lap, the agent receives a fixed reward, denoted by $C_c = 10$. This reward is designed to evaluate the performance of OPE methods in the presence of sparse events. Crucially, for the checkpoint reward to be well defined in s_t, a_t and s_{t+1}, we must assume that the already reached checkpoints are part of our state.

- **Lifetime Reward** The Lifetime Reward $R_t^{Lifetime}$ assigns a constant value of 1 to an agent at each time step where no termination is encountered. Hence:

$$R_t^{\text{Lifetime}}(s_t, a_t, s_{t+1}) = 1 - T_t(s_t) \tag{3.3}$$

  The Lifetime Reward is designed to evaluate the ability of OPE methods to predict when an agent will crash. Similar to the Checkpoint Reward, this reward investigates performance in the presence of sparse and relatively rare events. Although all our rewards incorporate sparse termination events, the Lifetime Reward specifically helps disentangle the influence of such events.

- **Minimum Action Reward** The Minimum Action Reward $R_t^{\text{Minimum Action}}$ is assigned based on the absolute steering angle at each time step. Formally, let steer be the absolute steering angle at time step t and $\text{steer}_{\max}$ be the maximum steering angle. The Minimum Action Reward is defined as:

$$R_t^{\text{Minimum Action}}(s_t, a_t, s_{t+1}) = 1 - \left(\frac{a_t^{\text{steer}}}{\text{steer}_{\max}} \right)^2 \tag{3.4}$$

  The Minimum Action Reward is high if there is little steering away from the neutral position and low if the steering angle is large. This reward is always between 0 and 1. While our previous rewards are defined over the state dimensions, this reward is defined over parts of the action space to investigate performance of OPE with respect to rewards defined over the action space.

Termination Condition and Truncation

All trajectories are truncated after 250 timesteps, corresponding to an episode covering 12.5 seconds if no crash occurs. The truncation length has been chosen so that fast agents can complete a bit more than a lap and recording a large number of trajectories is still feasible.

Intuitively, we wish to terminate a trajectory if our agent crashes into the boundaries of our track. To avoid excessive crashing in the real-world environment and damaging the hardware, we define a safety cone in front of the car that, if violated, sends a breaking command and terminates the current trajectory.

Assessing a safety cone breach involves checking the LIDAR readings within a 50-degree arc to the left and right of the vehicle's forward direction. By calculating the cosine of each ray's angle multiplied by its length, the trajectory is terminated to prevent a collision if any value falls below 30 cm. The same termination condition is applied to data collected in the simulation environment to enable fair comparisons. In addition, a human operator can engage a safety termination manually in order to avoid hardware damage.

- **Termination Condition:** Formally, the condition is defined as:

$$T_t(s_t) = \begin{cases} 1 & \text{if } \cos(\theta_i) \cdot d_i < 0.3 \text{ for any } i \in [-50°, 50°] \\ 1 & \text{if } t > 250 \\ 1 & \text{if Human Operator Intervention} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

  where θ_i is the angle of the i-th LIDAR ray, d_i is its length and t the current timestep.

3.2 Agent Design and Parameterization

To investigate OPE methods, we collect data with a variety of agents. Past research has employed different strategies to come up with such a variety. [FNN+21] train different online RL algorithms, varying the seeds and picking evenly spaced snapshots during training as their agents. Similarly, [QZG+22] trains seven offline RL algorithms on their datasets, subsequently evaluating those. [VLA+21] considers two agent classes, with the first being essentially a random walker, synthesizing different agents by shifting the action probability distribution. For the second class, an ϵ-Greedy agent, a Q network is trained to predict the best action in a given state. Post-training, this agent is parameterized using various ϵ values, dictating how much random exploration the agent performs.

In this thesis, to avoid coping with sim-to-real transfer, we utilize two well-known driving controllers and parameterize them to achieve a variety of behaviors: a version of Follow-the-Gap (FTG), initially introduced by [SG12], and Pure Pursuit (PP), as introduced by [WST+85].

To fulfill the requirement for stochasticity in IS, inspired by [FNN+21], we introduce stochasticity to our action outputs. We add noise as a truncated normal distribution with a standard deviation of 0.03 to all our delta actions (as discussed earlier, our delta actions are capped at ±0.3). While a higher standard deviation leads to additional variance, our real-world experiments show that a high standard deviation leads to excessive crashing. This is mainly because the car's turning radius correlates with the standard deviation's magnitude.

The following Sections 3.2.1 and 3.2.2 explain the two utilized controllers in greater detail and discuss the different possible parameterizations. Our agents are implemented in numpy and support batches of input data for efficient parallel processing; this is necessary for efficient evaluation and training of methods like FQE. All agents in this section operate on the transformed observation dictionary, as discussed previously.

3.2.1 Follow-the-Gap

Follow-the-Gap (FTG), is a purely reactive F1TENTH racing method. Originally introduced by [SG12], it and its variants like the disparity extender [Ott19], have seen use in F1TENTH racing.

Steering Control: The guiding principle behind the FTG method is:

1. to identify a gap
2. steer towards the middle of the largest identified gap

This is accomplished using the LIDAR present on our racing actor. In our method, we first set a minimum horizon length h, identifying allowed steering angles as the angle of any LIDAR ray larger than h. We denote this set of viable, not blocked, steering angles with S_{nb}.

To avoid steering too close to a wall, LIDAR rays close to steering angles not in S_{nb} are removed from S_{nb}. The *Gap Blocker* parameter controls the number of removed rays. If S_{nb} is empty, we reduce our horizon and repeat this process until we have at least one ray in S_{nb}. If we find multiple, we pick the one with the largest range value and steer toward the middle of its gap. Figure 3.5 shows a visualization of inferring the steering angle in FTG.

Velocity Control: The target velocity for the vehicle is determined based on the length of the LIDAR ray located at the center of the FOV, adjusted by a factor, the Speed Division Coefficient. The process involves several steps: First, we clip the the middle ray to be at most the length of our Speed Division Coefficient, k. After dividing with the resulting value, which is between 0 and 1, we scale it with our maximum velocity.

The target velocity, v_{target}, is thus calculated using the following equation:

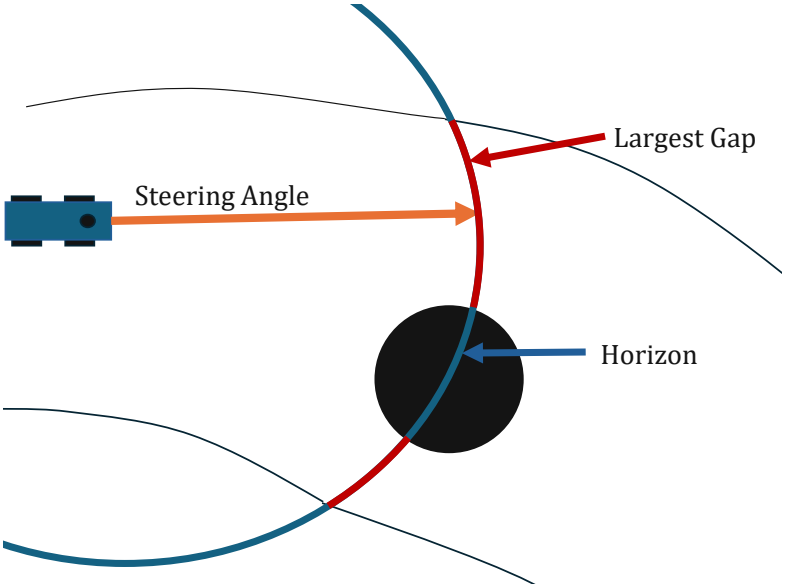


Figure 3.5: A visualization of FTG. The blue car is driving along a track and detects two gaps above and below an obstacle drawn in black. The algorithm picks the larger gap and steers towards its middle.

$$v_{\text{target}} = \left(\frac{\min(l_{\text{middle}}, k)}{k} \right) \times v_{\text{max}} \tag{3.6}$$

In this equation, l_{middle} represents the length of the central LIDAR ray, k is the Speed Division Coefficient, and v_{max} is the maximum speed of 5 meters per second. As for all agents, the minimum speed is set to 0.5 m/s.

The discussed procedure yields the parameters listed in Table 3.2 to adapt the behavior of our FTG algorithm.

Parameter Name	Utilized Range	Units
Gap Blocker	0-10	-
Initial Horizon	2-8	m
Speed Division Coefficient	0.4-1.2	-

Table 3.2: Adaptable parameters for different FTG agents.

We will now shortly discuss the influence of each parameter on agent behavior.

- **Initial Horizon:** The initial horizon significantly impacts agent behavior by defining how much the agent looks ahead in its decision-making. In general, short initial horizons lead to more suboptimal behavior. The horizon setting has no impact on the speed of the agent.

- **Gap Blocker:** This parameter also influences the steering behavior of the car and, crucially, how many horizon iterations have to be performed. With larger values leading to a shorter effective horizon. However, values that are too small can lead to collisions with the track boundaries.
- **Speed Division Coefficient:** The only parameter directly influencing the speed of the agent. Smaller values lead to higher speeds and larger values lead to lower speeds.

3.2.2 Pure Pursuit

Pure Pursuit (PP) is a geometric planner for trajectory tracking, first introduced by [WST+85]. In contrast with FTG, PP requires information on its current position on the racing track and needs a racing line consisting of multiple racing points.

Steering Controller: Given the current x and y position on the racing track, PP picks the next racing point that is in front of the current position at a certain lookahead distance. We calculate the steering angle as follows:

$$\delta = \text{atan}\left(\frac{2 \times l_{wb} \times \sin(\alpha)}{l}\right) \tag{3.7}$$

Where α is the angle from the driving direction axis of the vehicle to the racing point. The wheelbase, l_{wb}, the distance between the front and rear axles of the car, set to 0.3 meters in our case, and l the lookahead distance. Calculating the steering angle in this way, as opposed to simply picking α as our target, leads to driving on an arc, resulting in more stable behavior. A visualization of PP steering is given in Figure 3.6.

Velocity Controller: The agent's speed is inferred according to a precomputed speed profile of the raceline, stored in the raceline points, and scaled by the *Speed Scaling Coefficient*.

Therefore, we have the parameters listed in Table 3.3 to adapt our PP algorithm.

Parameter Name	Utilized Range	Unit
Raceline File	1-8	-
Speed Scaling Coefficient	0.3-1.0	-
Lookahead distance	0.5 -1.4	m

Table 3.3: Adaptable parameters for different PP agents.

In the following, we briefly describe each parameter in more detail.

- **Raceline File:** We generated eight different racelines by utilizing the TUM global race trajectory optimization tool chain [TUM23]. We generate diverse racelines by modifying the original racetrack image (e.g. adding additional obstacles) and varying various parameters within the toolchain. Figure 3.7 shows the different generated racelines. Which influences the car's speed and behaviors.

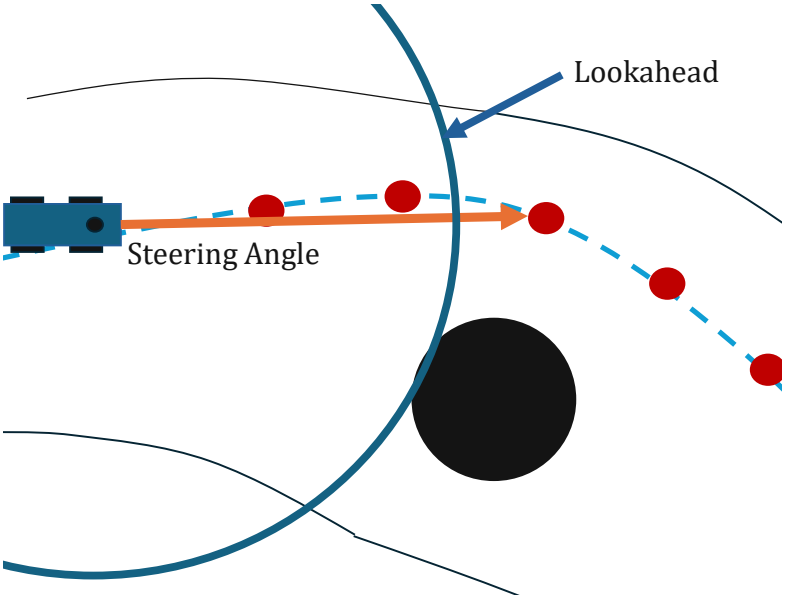


Figure 3.6: A visualization of PP. The blue car is driving along a track and directly steers towards the first raceline point after the lookahead distance. This visualization shows α as the steering angle for simplicity and visual clarity.

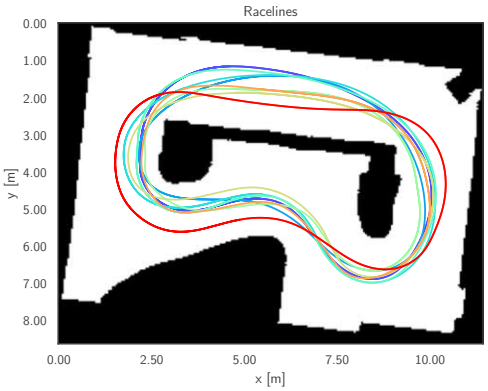


Figure 3.7: Different racelines used for PP agents. The driving direction is clockwise.

- **Speed Scaling Coefficient:** We scale the velocity targets of the raceline with this coefficient. High-speed coefficients may reduce the trajectory tracking capabilities and are subject to crashes.
- **Lookahead Distances:** Changing the lookahead distance can significantly influence the agent's behavior. Picking a smaller lookahead makes the agent more unstable in the real world due to the delay in commands and estimation. A large lookahead results in the agent cutting corners.

3.3 Real-world Dataset Collection Protocol

Since OPE methods always predict performance with respect to starting points in the state space, and we wish to evaluate other agents from the same starting points, we pick a total of 10 possible starting points along the centerline. Our *experiment controller* automatically starts rollouts from these starting points and reduces the amount of human oversight needed. The experiment controller observes the following protocol:

1. In the initial state, the agent is dormant. The agent does not move but can be steered via remote commands by a human operator. The human operator has to initiate step 2 manually.
2. Pick and remove a starting point from our list of remaining starting points that is relatively close to our current position but sufficiently far away such that we can easily reach a stable state when following the centerline. If no new starting points are available, restart with all original starting points.
3. Navigate with PP along the centerline until within <1m of the starting point.
4. Reduce speed to 0.5 m/s and set steering to neutral for 1 second.
5. Engage the agent and start logging the observations provided to the agent and the actions emitted.
6. Terminate the trajectory if (a) or (b)
   a) If the human operator terminates the trajectory or our termination condition is met, we terminate the trajectory and enter the dormant state - human intervention is needed.
   b) If the trajectory reaches the truncation length - set to 250 timesteps - we return to 2.; starting initialization of the next trajectory rollout.

Crucially, in step 4 we do not fully stop the car, since starting the motor from completely idle is sometimes associated with random delays in our hardware and software setup. After we have collected a sufficient amount of trajectories for our target agent we update our agent configuration and start the next set of rollouts with the next agent. Note that we collect the same amount of trajectories for each of our ten initial starting states.

3.4 Real-World F1TENTH Offline Dataset

The resulting real-world F1TENTH dataset comprises 1305 trajectories from 55 unique agents. Among them, 51 agents contributed 10 or more trajectories each. It is important to note that there is a small variation in trajectory counts for some agents due to manual data cleaning, which removed several trajectories for reasons such as inaccurate state estimation at the trajectory's start or improper manual termination.

The dataset is divided into a training and evaluation set. The training set consists of 480 trajectories collected with 40 unique agents, with 36 agents each contributing 10 to 20 trajectories. The evaluation set includes 825 trajectories from 15 different agents, where each agent's contribution ranges from 48 to 90 trajectories, though most agents are represented with exactly 50 trajectories. Figure 3.8 shows the trajectories in the training and evaluation set in the x/y position space. Table 3.4 shows various statistics on the evaluation and training dataset.

	Training	Evaluation	Total
Nr. Agents	40	15	55
FTG	8	4	12
PP	32	11	43
Nr. Trajectories	480	825	1 305
Nr. Terminations	285	417	702
Nr. Transitions	79 395	148 529	227 924

Table 3.4: Composition of the Real-World F1TENTH dataset.

As we are in a continuous and stochastic environment, the starts of our trajectories are clustered around our chosen starting points. Figure 3.9 shows the actual starting points of all trajectories in our dataset. It can be observed that they are indeed clustered heavily around ten starting points, but some outliers exist.

Figures 3.10-3.13 plot the discounted sum of rewards for the evaluation and training sets, for the different rewards discussed in Section 3.1.2. The ordering of the agents is kept the same in all plots. It is important to note that a large part of the variance visible for both the training and evaluation agents is due to the ten different starting positions.

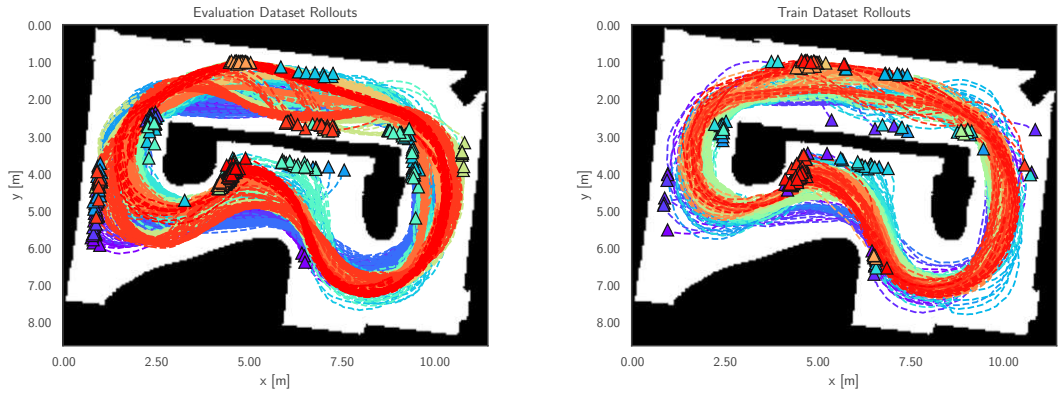


Figure 3.8: Visualization of evaluation and training rollouts in the x/y observation space. Terminations are marked with a Δ. Different colors represent different agents.

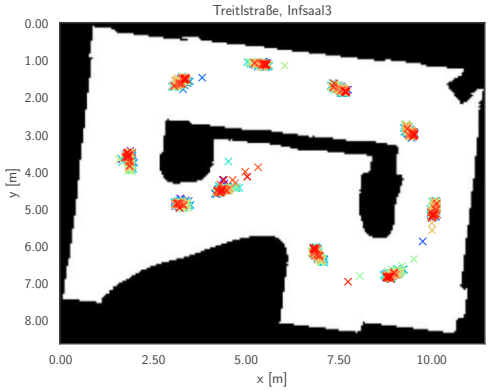


Figure 3.9: Our map with starting points.

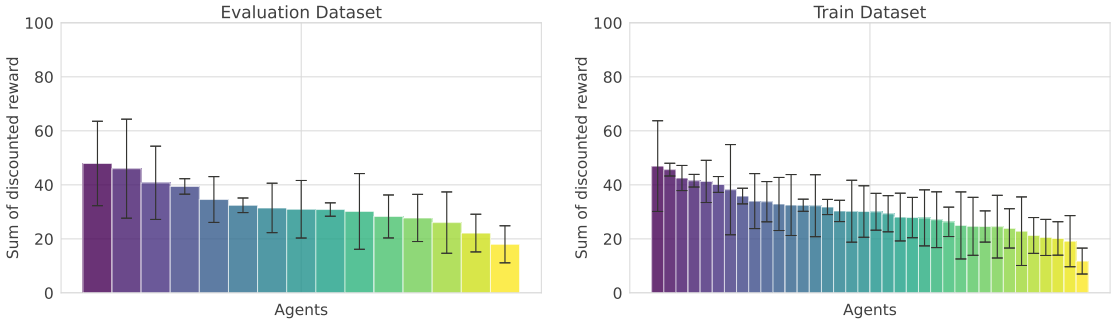


Figure 3.10: Performance as measured by the discounted sum of the Progress Reward with $\gamma = 0.99$. The left figure shows evaluation agents; the right figure training set agents.

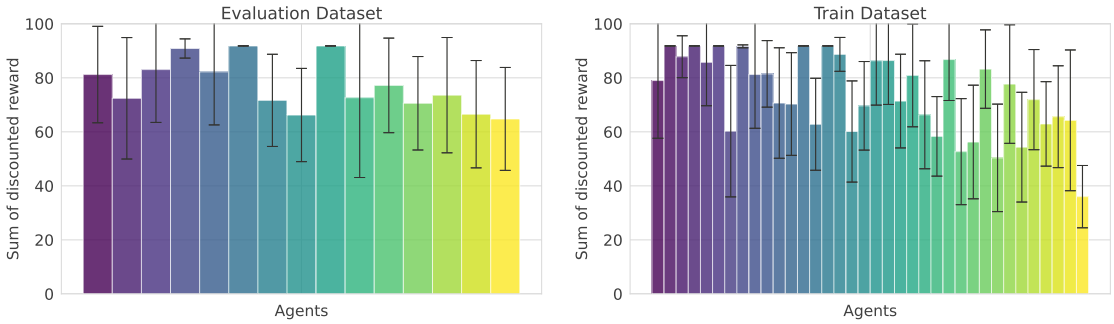


Figure 3.11: Performance as measured by the discounted sum of the Lifetime Reward with $\gamma = 0.99$. The left figure shows evaluation agents; the right figure training set agents.

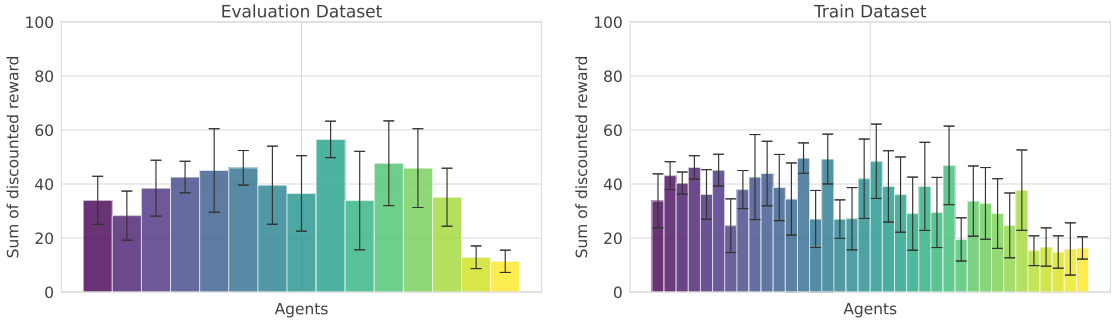


Figure 3.12: Performance as measured by the discounted sum of the Minimum Action Reward with $\gamma = 0.99$. The left figure shows evaluation agents; the right figure training set agents.

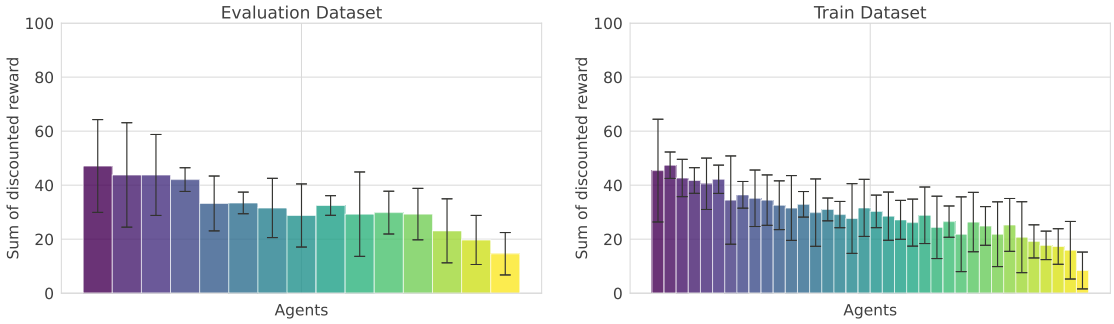


Figure 3.13: Performance as measured by the discounted sum of the Checkpoint Reward with $\gamma = 0.99$. The left figure shows evaluation agents; the right figure training set agents.

CHAPTER 4

Deep Off-Policy Evaluation for Autonomous Racing Cars

This chapter discusses the Off-Policy Evaluation methods investigated in this thesis, focusing on adaptations from previous works to the F1TENTH environment. Section 4.1 describes the Model-based methods explored in our research. Section 4.2 discusses FQE methods. Section 4.3 examines additional IS methods that were not already introduced, and also introduces our novel TPDWIS estimator. Finally, Section 4.4 discusses the DR estimators investigated.

4.1 Model-based

We investigate a total of eight distinct Model-based approaches. Due to the popularity of simulation as an OPE method, we first briefly discuss the `f1tenth-gym` simulator. The remainder of the section focuses on learning-based methods. We describe the training details pertinent to all trained models in Section 4.1.2. Section 4.1.3 addresses adaptations of the state-space to facilitate training, ensuring that the models can effectively learn from the data. Section 4.1.4 discusses the loss functions employed for optimizing our learning-based models, Log-Likelihood and Mean-Squared-Error. Section 4.1.5 discusses the various models and architectures, whose performance we investigate. Finally, Section 4.1.6 discusses the evaluation procedure for MB methods. Table 4.1 presents an overview of the various combinations of state-space models and loss functions utilized in this thesis.

4.1.1 Simulation

We utilize the simulator from [OZKM20], with slight modifications to allow starting at a speed of 0.5 m/s, consistent with our dataset trajectories. Studying the performance of the simulator to provide OPE estimates is crucial for comparing the ability of OPE to bridge

Model Name	Loss Type	From
Simulation	-	[OZKM20]
Naive Model (MSE)	MSE	[KN20]
Naive Model (LL)	LL	[ZPN+21]
Delta Model (MSE)	MSE	[Far23]
Delta Model (LL)	LL	this thesis
Autoregressive Model	LL	[ZPN+21]
Autoregressive Delta Model	LL	this thesis
Ensemble (any model)	MSE or LL	this thesis

Table 4.1: Overview of the models and employed loss functions investigated in this thesis. The Ensemble Model can be combined with any of the other models to enhance performance.

the sim-to-real gap. The default parameters of the simulator are employed, replicating the experience of a typical user. For more detailed information on the simulator, readers are referred to [OZKM20].

4.1.2 Common Training Procedure

In order to fit our dynamics model $\mathcal{F}$ we sample one transition (or rather a batch of transitions) from our training dataset $(s_t, a_t, s_{t+1}) \sim \mathcal{D}_b$. We then transform the sampled values as described in Section 4.1.3 and provide them to our neural network (see Section 4.1.5 for more details on the architectures employed), performing a forward pass followed by the loss computation, as described in Section 4.1.4. Subsequently, we train the neural network parameters with backprogation.

We use the Adam optimizer to train all models, with a learning rate of 0.001 and weight decay of 0.0001. Training proceeds for 10,000 steps, chosen based on the stabilization of our evaluation metric—the sum of mean squared errors (MSE) between predicted and actual state trajectories. The training set is shuffled, and we employ a batch size of 256. The training was conducted using PyTorch version 2.1.1 on an NVIDIA GeForce RTX 3090.

4.1.3 Observation and Action Space Transformations

Section 3.1.2 discussed the observation dictionary provided to our agents that is also logged[1]. To reduce complexity in learning the state transition function $\mathcal{F}$, we opt for multiple transformations of the original observation space. These transformations are crucial for improving the efficiency and accuracy of our models. We apply the following transformations before passing data to $\mathcal{F}$:

[1]Additional data is logged as well and available in our provided dataset; however, for this thesis, we will rely on information available from the observation dictionary.

Normalization: We normalize all state dimensions using the mean and standard deviation of the associated state dimensions in the training set.

LIDAR Ranges: There are a total of 54 LIDAR ranges in our single timestep observation dictionary. Since the LIDAR rays do not influence the dynamics of the underlying system, we remove them from the state-space. However, since our agent requires LIDAR rays to infer the next action (FTG does), we compute the next timestep LIDAR rays with a function $O(s'_x, s'_y, s'_\theta)$, based on the predicted position values. For our implementation of O, we perform raycasting on our map, as implemented in the simulator introduced earlier [OZKM20]. For future approaches that do not want to rely on such hand-crafted functions, it could be feasible to learn O from the dataset, for example, as a neural network.

Cyclic encoding of θ: To facilitate smoother learning, we transform the rotation with respect to our map θ (originally ranging from 0 to 2π) into its sine and cosine components. This commonly used encoding enhances the learning process by explicitly encoding the equivalence of 0 and 2π, avoiding discontinuities.

Previous Actions: The previous actions are included in our original observation dictionary. However, since they are not dependent on the dynamics of the system we remove them from the state vector provided to our MB method. Instead, we use them to compute the raw action a, as the previous action plus the delta action, that we provide, normalized, to $\mathcal{F}$.

Hence, the states s that we provide to our dynamics function, and which we receive back, consist of the observations listed in Table 4.2.

Observation	Unit (when not normalized)
position x	m
position y	m
position θ (sin)	rad
position θ (cos)	rad
linear velocity x	m/s
linear velocity y	m/s
angular velocity z	m/s

Table 4.2: The observations provided to our MB dynamics method.

4.1.4 Loss Functions

To train the neural networks described in the rest of this section, we require a divergence measure between the predictions and our targets: the loss function. The batch Mean-Squared-Error (MSE) loss function is defined as:

$$\text{loss}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(s_{t+1,i} - \mathcal{F}(s_{t,i}, a_{t,i}))^2 \tag{4.1}$$

where the additional subscript i indicates the index of the states and actions in our training batch.

Training the model in a probabilistic fashion with a Log-Likelihood (LL) loss makes it possible to capture uncertainty in the state transitions. Training models this way has shown promising results in previous studies [ZPN+21, LKTF20]. To train our network in such a way, the network's output must be adapted. Instead of returning a singular prediction for the next state, the network must return a probability distribution. Generally, in previous works, a multivariate normal distribution is returned by the network (e.g., [ZPN+21] and [LKTF20]).

To return a multivariate normal distribution, the network is adapted to output the mean (μ_{t+1}) and the logarithmic standard deviation ($\log \sigma_{t+1}$) of the predicted distribution for each next state dimension:

$$\mu_{t+1}, \log \sigma_{t+1} = \mathcal{F}(s_t, a_t) \tag{4.2}$$

Any given model $\mathcal{F}$ can be adapted to return such a distribution by doubling the size of the output layer. The Log-Likelihood loss is then given as:

$$\text{loss}_{\text{LL}} = \sum_{i=1}^{n} -\log \mathcal{N}(s_{t+1,i} | \mu_{t+1,i}, \sigma_{t+1,i}) \tag{4.3}$$

As with the MSE loss, we sum over the batch size, with i denoting the index in the batch. The distribution $\mathcal{N}(s_{t+1,i} | \mu_{t+1,i}, \sigma_{t+1,i})$ represents the multivariate normal distribution estimating the likelihood of $s_{t+1,i}$ given our model outputs. When evaluating our estimator according to Equation 2.5, we discard the standard deviation and perform next state prediction based only on the mean.

4.1.5 Model Architecture

The various state-space models from Table 4.1 are discussed here. These approaches are based on the works of [ZPN+21], the Highway Environment [Far23], and [KN20], or are novel combinations of these models and loss functions, as indicated in the table.

Naive Model (MSE & LL)

The naive model, as presented in the works of [KN20] and [ZPN+21] in the OPE context, is a simple feed-forward neural network that fits $\mathcal{F}$ from Equation 2.4 in a straightforward, supervised manner. The input of the neural network is assembled by concatenating the state and action, and the next state prediction is computed by performing a forward pass through the network.

During training, a batch of associated states, actions, and next states is sampled. The states and actions are then passed through the network to compute predictions for the next states. The loss function is applied to our predictions and target next state

values. Subsequently, back-propagation through the neural network is performed, and the network parameters are updated according to the gradients and our optimizer settings.

The architecture of our neural network is equivalent to that adopted by [KN20]. After concatenating our states and actions, they are passed through four hidden layers, each comprising 256 units, culminating in an output layer that delivers a tensor matching the dimensionality of the state-space. The ReLU activation function is employed after each layer, except for the output layer, to ensure non-linearity in the model's computations. The LL version of the naive model is equivalent to the MSE version, with the difference that the output size is doubled, as discussed in Section 4.1.4.

As the original implementation is only available in TensorFlow, we ported the code to PyTorch.

Delta Model (MSE & LL)

The naive model offers several ideas for improvement, especially if more context is known about the specific dynamics problem we want to learn. While not yet evaluated on a previous OPE benchmark, the implementation of a MB method in the Highway Environment for autonomous driving [Far23] is relevant to our problem setting.

Motivated by classical control theory, [Far23] reformulates our next state prediction as follows:

$$\begin{aligned} dx &= A(\tilde{s_t}, a) \cdot s_t + B(\tilde{s_t}, a) \cdot a \\ s_{t+1} &= s_t + \Delta t \cdot dx \end{aligned} \tag{4.4}$$

where $\tilde{s_t}$ represents the modified state vector with the x, y, and θ components set to zero to eliminate direct dependencies on position and orientation. In [Far23] only the x and y components are set to zero; we additionally set θ to zero since the current rotation is also invariant to the delta next state. This modification helps increase sample efficiency by ensuring the model focuses on learning the underlying dynamics independent of the specific location and orientations.

A and B are neural networks we fit during training with backpropagation. A outputs a matrix with dimensions equal to the state-by-state dimension. The output layer of B produces a matrix of state-by-action dimension. Since our θ is split into sine and cosine components, we additionally normalize these components to ensure the trigonometric identity $1 = \theta_{\sin}^2 + \theta_{\cos}^2$ is fulfilled for our next state s'. Lastly, $\Delta t = \frac{1}{20}$, which is the timestep size in our environment.

To train the Delta Model with LL loss, we double the size of dx from Equation 4.4 and correspondingly increase the size of the matrices returned by A and B. Subsequently, we use the second half of dx as our log standard deviations to train with LL loss.

We re-implement and adapt the available code from [Far23] in PyTorch. The Delta Model is expected to improve performance by focusing on the relative changes in state.

Autoregressive Model

Autoregressive Models (AMs) have shown promising results when employed for OPE on the RL Unplugged Benchmark [ZPN+21]. The advantage of such models lies in their ability to capture dependencies between the state dimensions, which can lead to more accurate predictions. Instead of directly predicting the next state in a single forward pass like the previous models, the next state is constructed iteratively by predicting subsequent state dimensions. Each subsequent state dimension prediction is conditioned on the already predicted state dimensions.

During the training process, the loss is accumulated across each state dimension, and backpropagation is performed after the entire next state has been predicted. Pseudo-code detailing the Autoregressive Model's forward pass and training update is presented in Algorithms 4.1 and 4.2, respectively.

Algorithm 4.1: Autoregressive Model Prediction

Input: State vector s, action vector a, model θ
Output: Next state vector s'

$s' \leftarrow \text{ZeroTensorLike}(s)$ `// Initialize` s'
for $i \leftarrow 1$ **to** *size(s)* **do**
  $s\_one\_hot \leftarrow \text{OneHot}(i, \text{size}(s))$ `// One-hot encode dimension` i
  $input\_states \leftarrow \text{Concatenate}(s, s', s\_one\_hot)$
  $s'[:, i] \leftarrow \theta(input\_states, a)$ `// Update dimension` i
end
return s';

As a base model, we utilize the naive model. In accordance with [ZPN+21] we make the following changes to make our model autoregressive: The dimensionality of the input is changed to be three times the original state dimension. This accommodates the addition of the next state tensor into the model's input and a one-hot encoding vector to inform the model of the next state dimension to predict in this forward pass. Secondly, the network output is changed to provide two values, the prediction of the dimension of the next state and the log standard deviation, for training with LL loss.

Autoregressive Delta Model

For this model, we combine the AM and Delta Model. In the implementation of this model, we employ the Delta Model (LL) and adapt the input of our A and B networks to $\times 3$ the original size, as similarly done for the single network in the Autoregressive Model. We then employ a wrapper around this model that only returns one state at a time to reproduce the functionality of an AM. When predicting the heading θ, we return the sine and cosine components jointly to maintain the trigonometric identity. Additionally, we return the relevant log standard deviation as computed by the DM for training with LL loss.

Algorithm 4.2: Autoregressive Model Update Step

Input: State vector s, action vector a, next state vector s', model θ
Output: Updated model θ

begin
  Initialize $total\_loss \leftarrow 0$;
  for $i \leftarrow 1$ **to** *size(s)* **do**
    $s\_one\_hot \leftarrow \text{OneHot}(i, \text{size}(s))$;
    `// One-hot encode dimension` i
    $s\_next\_state \leftarrow s'$ with elements from i onwards set to 0;
    $target \leftarrow s'[i]$;
    $input\_states \leftarrow \text{Concatenate}(s, s\_next\_state, s\_one\_hot)$;
    $prediction \leftarrow \theta(input\_states, a)$ `// Model prediction for dimension` i
    $loss \leftarrow \text{LossFunction}(target, prediction)$;
    `// Compute loss for dimension` i
    $total\_loss \leftarrow total\_loss + loss$;
  end
  $\theta \leftarrow \theta - \alpha \nabla_\theta total\_loss$;
  return θ;
end

By combining the iterative prediction capability of AM with the prediction of state changes of DM, the ADM model is expected to achieve more accurate and robust predictions of the next state. This integration allows the model to capture dependencies between state dimensions while focusing on the underlying dynamics independent of specific positions and orientations.

Model Ensembles

It is well established that combining the predictions of multiple models can improve overall performance. This approach can potentially decrease variance in the output prediction and smooth out errors. One way to train such multi-model ensembles is by resampling with replacement from our original batch, generating slightly different training batches for each model in the ensemble. Inference is then performed by averaging the output of all models across all state dimensions.

We investigate the performance of ensembles of the Naive Model (MSE and LL), AM, and the ADM. Each ensemble consists of five models. We will denote a specific ensemble of a model with the model name followed by the number five in brackets (e.g., NM(5), AM(5), ADM(5)).

4.1.6 Evaluation Procedure

As discussed in Section 3.4, the starting states of our evaluation trajectories do not completely align with our training starting states due to the continuous nature of our environment. To provide the best possible estimations of the evaluation trajectories, we start our MB trajectory rollouts from the recorded initial states of the evaluation trajectories associated with the target agent. We perform one trajectory rollout per starting state, resulting in approximately 50 trajectories per agent. Termination detection is performed in accordance with the ground-truth termination function outlined in Section 3.1.2.

4.2 Fitted Q-Evaluation

Following previous work, we investigate two different versions of FQE [FNN+21, ZPN+21]. These two versions differ in the output of the Q function, being either a scalar return value, as introduced in the seminal work by [LVY19], or a probability distribution [PPM+20, BMHB+18].

First, Section 4.2.1 discusses common training settings. Section 4.2.2 discusses the observation and action space transformations we employ to enable efficient training. Section 4.2.3 discusses the loss function, the most significant differentiator between our two employed versions of FQE. In Section 4.2.4 we briefly discuss the neural network model. Last, we discuss the evaluation procedure we adopt in Section 4.2.5.

4.2.1 Common Training Procedure

We sample a transition from our shuffled training dataset, this time to learn our Q-function: $(s_t, a_t, s_{t+1}, r_t) \sim \mathcal{D}_b$. Note that, contrary to MB, we also include the single-step reward r_t in our sample, as we need it to compute the Bellman backup. Next, we transform our state and actions as outlined in Section 4.2.2. Following the training procedure in Algorithm 2.1, we then compute our targets and predictions computing the loss as outlined in Section 4.2.3.

We use the Adam optimizer to train all models, setting a learning rate of 0.0001 and weight decay of 0.00001. Training proceeds for 200,000 steps, chosen based on observed convergence and a limited compute budget. Details on the performance of FQE across the training timesteps are given in Section 5.2.2. The training set is shuffled, and we employ a batch size of 256. Our implementation is based on the TensorFlow version of FQE-L2 from [KN20]. Therefore, as in standard RL Q-learning, we utilize a second target network to reduce the effect of a shifting target function and to stabilize training. The training was conducted using PyTorch version 2.1.1.

4.2.2 Observation and Action Space Transformations

In addition to the transformations of the data described in Section 4.1.3, we add a timestep observation, $t_{obs} \in (0, 1)$. The need for this additional observation is evident when considering that without t_{obs}, there is no information in the sampled state indicating how soon the trajectory will be truncated.

4.2.3 Loss Function

FQE-L2 This version of FQE minimizes the L2 error of the target values. Therefore, the loss function for a batch of data with size N is given as follows:

$$\text{loss}_{\text{L2}} = \sum_{i=1}^{N} (y_{(i)} - \hat{y}_{(i)})^2 \tag{4.5}$$

where y is obtained with a single-step Bellman backup, as seen in line 4 of Algorithm 2.1, and $\hat{y}$ is our direct Q-prediction.

Furthermore, we scale the direct output of the Q network by multiplying it with 100. Since $100 > R^\tau > 0$, for all of our trajectories and rewards, the target output of the Q network should be between 0 and 1. This approach is inspired by code provided by [KN20].

FQE-DD [PPM+20] utilize a discrete distributional Q-critic network. This method extends the concept of scalar return values to predict entire discrete probability distributions, as originally introduced to RL with the C-51 network architecture proposed by [BDM17].

For the implementation of FQE-DD, additional hyperparameters, dictating the properties of the discrete output distribution have to be defined. First, specifying the range of possible rewards, denoted by r_{min} and r_{max}, to define the bounds of the Q-function's output distribution is necessary. Additionally, the discrete output distribution's granularity is determined by the output layer's size l. In this thesis we choose $r_{min} = 0$, $r_{max} = 100$ and $l = 101$.

The output distribution of the Q-network is then constructed by mapping each output neuron $o_0, \ldots, o_{l-1}$ onto our discrete distribution. With j being the index of the output neuron, the mapping is straightforward:

$$r_j = r_{min} + j \frac{r_{max} - r_{min}}{l - 1} \tag{4.6}$$

For each output atom, we compute an associated probability p_i by taking the exponential, ensuring normalization over all outputs:

$$p_i = \frac{\exp(o_i)}{\sum_{j=0}^{l-1} \exp(o_j)}. \tag{4.7}$$

Together with the computed locations in the distribution, the probabilities yield a discrete output distribution, assigning a (normalized) probability p_i to each possible reward r_i in the output distribution.

As discussed in [BDM17], training the discrete distribution network is not entirely straightforward, as the computed target y from Algorithm 2.1 in line 4 does not necessarily coincide with the support of the output distribution. To train the model, the probability mass of each result of the Bellman backup is distributed to the closest support values in proportion to the distance to the support atoms. This method allows our model to handle the potentially continuous values yielded by the Bellman backup and be trained with standard categorical cross-entropy:

$$\text{loss}_{\text{CE}} = -\sum_{i=1}^{N}\sum_{j=1}^{l} y_{(i,j)} \cdot log(\hat{y}_{(i,j)}) \tag{4.8}$$

Here, both $y_{(i,j)}$ and $\hat{y}_{(i,j)}$ are represented as vectors of length l, encapsulating the distribution across discrete reward bins. The subscript j denotes the jth bin in this distribution, to which the Bellman backup target $y_{(i,j)}$ has been appropriately discretized following the method outlined above. For more details, the reader is referred to [BDM17].

4.2.4 Model Architecture

We employ the neural network model as for the naive MB method described in Section 4.1.5. We adjust the input size to fit the expanded state-space for FQE, and we adapt the output layer in accordance with the loss function employed.

4.2.5 Evaluation Procedure

The evaluation procedure for FQE is equivalent to the one employed for MB. However, instead of performing rollouts, we query the Q-function with multiple sampled actions according to the initial evaluation state. To estimate the expectation, as described in Equation 2.6, we sample twenty actions per initial state.

4.3 Importance Sampling Methods

As Importance Sampling does not utilize any function approximations and differs significantly from the two previously discussed estimator families, this section exhibits a different structure. First, Section 4.3.1, motivates IS as a valid approach in our environment. Then Section 4.3.2 provides general observations on the IS ratio, particularly for our environment, and motivates our novel estimator introduced later. Section 4.3.3 and 4.3.4 discuss Per-Horizon Weighted Importance Sampling (PHWIS) [DTB18] and Per-Decision-Weighted Importance Sampling (PDWIS) [Tho15] respectively. Based on these two estimators, Section 4.3.5 introduces the novel Termination-aware Per-Decision-Weighted Importance Sampling (TPDWIS) estimator that extends PDWIS to handle

variable-length trajectories in a statistically consistent way. Furthermore, we provide a proof of the consistency of the TPDWIS estimator.

4.3.1 Validity of Importance Sampling in F1TENTH

Let us now briefly reconsider Assumption 2, namely $\pi(a_t|s_t) \neq 0 \implies \pi_b(a_t|s_t) \neq 0$, made in Section 2.4.3 for IS to be valid in a MDP. In our case, the two distributions are the target and behavior agent action distributions. Since we, as discussed in Section 3.1.2, employ truncated normal distributions that are defined over the entire feasible action space (truncation only occurs at the boundaries where actions are not possible), it is guaranteed that for all our i behavioral agents $\pi_i(a_t|s_t) > 0$ holds, and the assumption is therefore always satisfied, leading to IS techniques being valid in our setting. Therefore, we can employ IS, WIS and PDIS as discussed in Section 2.4.3.

4.3.2 Importance Sampling ratio for long time horizons

As discussed in Section 2.4.3, the IS ratio lies at the core of all our IS-based methods. Hence, we make some general observations about the IS ratio in our environment here. Figure 4.1 showcases the trend of the cumulative IS ratios $p_{0:t}$ by plotting the ratios across time steps t for various behavior agent trajectories, under one specific target policy. Each line represents a different trajectory, colored differently to distinguish between agents with varied parameterizations.

Two key observations can be made from Figure 4.1: First, the rate at which the cumulative product of the IS ratios decreases varies among agents with different parameterizations. For instance, the green trajectories, associated with one specific behavior agent, show a slower rate of decrease compared to the darker blue trajectories, which correspond to a different agent. Trajectories, and hence also agents, with more gradually decreasing IS ratios can be considered to be more aligned with our target agent, since the probabilities of taking the same actions are higher.

Second, consistent with the observations by [SMGDV21] and [DTB18], the magnitude of the trajectory-wise cumulative IS ratio is linked to the episode's length. The prominent $\times$ on the top left of Figure 4.1, marking an early crash by the blue agent, exemplifies this. The cumulative IS ratio associated with this trajectory is notably higher than that of all other trajectories due to the early termination of the episode. Nonetheless, examining the long-term IS ratios for this agent across different rollouts reveals a rapid decrease, underscoring a significant behavioral deviation from our target agent. Indeed the blue agent is a FTG agent, while our target agent is a PP agent.

This difference in magnitude of the cumulative IS ratios for different length trajectories presents a challenge. Estimators such as IS and WIS, which rely solely on trajectory-wise IS ratios, will assign disproportionately large weights to early terminating trajectories. Consequently, this leads to an underestimation of the true reward.

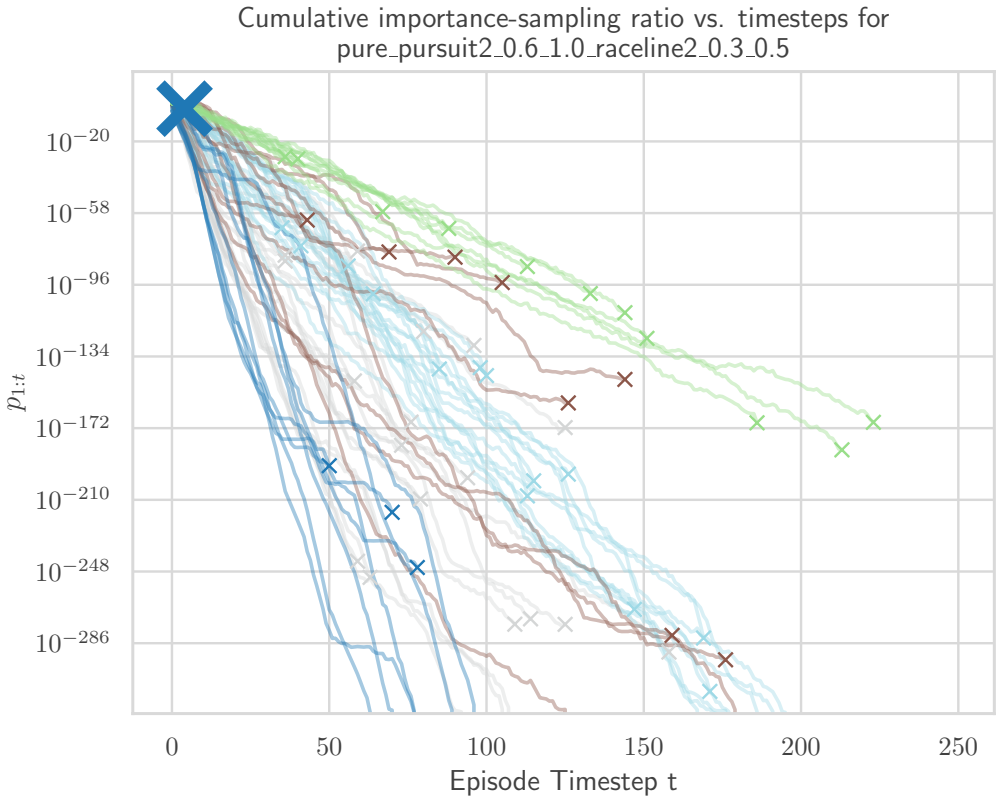


Figure 4.1: Cumulative IS ratios across trajectories. Note the logarithmic scale on the y-axis. Each × marks the end of a trajectory, representing the final cumulative IS ratio of the trajectory. Different colors denote trajectories from different agents.

4.3.3 Per-Horizon Weighted Importance Sampling

In order to solve the issue of different length trajectories [DTB18] propose the Per-Horizon Weighted Importance Sampling (PHWIS) estimator. Their estimator is given as:

$$V_{PHWIS} = \sum_{t=1}^{\max_\tau T(\tau)} W_t \sum_{\substack{\tau \in D_b \\ T(\tau)=t}} \gamma^{t-1} r_t^\tau \frac{p_{1:t}^\tau}{\sum_{\substack{\tau \in D_b \\ T(\tau)=t}} p_{1:t}^\tau} \tag{4.9}$$

where

$$W_t = \frac{\sum_{\tau \in \mathcal{D}_b} \left[\mathbf{1}_{\{T(\tau)=t\}} p_{1:t}^\tau \right]}{\sum_{\tau \in \mathcal{D}_b} p_{1:min(t,T(\tau))}^\tau} \tag{4.10}$$

In this approach, the WIS ratio is computed for each time horizon separately and subsequently reweighted. However, this approach still leads to the importance of shorter

trajectories being severely overestimated since W_t retains the property of assigning significantly higher weights to shorter trajectories [DTB18]. This is the case because of the minimum we take in the denominator of W_t; a large weight due to early termination will dominate later timesteps in the denominator. Hence [DTB18] propose a second estimator, Heuristic Per-Horizon Weighted Importance Sampling (HPHWIS):

$$W_t^h = \frac{\sum_{\tau \in \mathcal{D}_b} \left[\mathbf{1}_{\{T(\tau)=t\}} p_{1:t}^{\tau} \right]^{\frac{1}{T(\tau)}}}{\sum_{\tau \in \mathcal{D}_b} p_{1:min(t,T(\tau))}^{\tau}} \tag{4.11}$$

In cases where the expression $\mathbf{1}_{\{T(\tau)=t\}} p_{1:t}^{\tau} < 1$ (which is empirically the case in many settings) this corresponds with an increase in magnitude for longer horizons. While they obtain good empirical results in their work on toy examples, it is unclear if even the unmodified V_{PHWIS} is a statistically consistent estimator.

4.3.4 Per-Decision-Weighted Importance Sampling

As pointed out in Section 4.3.2, only considering the cumulative IS ratio can lead to underestimation. Hence, we might expect better performance on our data with PDIS. However, as can be observed in Figure 4.1, our IS ratios are very small, leading to severe underestimation if not utilizing some form of normalization.

The concept of a Per-Decision-Weighted Importance Sampling (PDWIS) estimator originates from [PSS00]. However, their version is inconsistent as shown in [Tho15]. [Tho15] introduces a consistent version of PDWIS. For brevity, we will refer to this version as PDWIS in the remainder of this work. Similar to WIS, PDWIS is no longer unbiased, but consistent [Tho15].

To formulate the PDWIS estimator, we adjust our importance weights at each timestep so their sum equals one. Hence, the PDWIS estimator is given as:

$$V_{PDWIS}^{\pi} = \sum_{t=1}^{\max_\tau T(\tau)} \sum_{\forall \tau \in D_b} \gamma^{t-1} r_t^{\tau} \cdot \frac{p_{1:t}^{\tau}}{\sum_{\tau \in \mathcal{D}_b} p_{1:t}^{\tau}} \tag{4.12}$$

By normalizing weights at each timestep, the maximum (or minimum) return is effectively represented by the sum of the maximum (or minimum) rewards observed at each timestep across all considered trajectories.

The PDWIS estimator raises a crucial question about managing trajectories that end prematurely. Since our calculation extends over the length of the longest trajectory in our dataset, how should we adjust the IS ratio, $p_{\tau,1:t}$, for those trajectories that conclude before reaching time t? This question is discussed in the remainder of this section and culminates in the introduction of our novel TPDWIS estimator.

PDWIS (Constant)

In the Caltech OPE Benchmarking Suite (COBS) [VLA+21], the authors set the timestep-wise IS ratio $p_t^\tau = 1$ for $T(\tau) < t$. This is equivalent to the minimum operation employed in PHWIS.

Clearly, this approach suffers from the previously discussed drawbacks: since we set all timestep-wise IS ratios to one after a crash, the cumulative IS ratios of early crashes will eventually dominate our normalized IS ratio, being only close to one where the step-wise reward is zero (a trajectory that has crashed previously) and very close to zero elsewhere.

We will denote this version with PDWIS (C).

PDWIS (Mean)

The observation from Section 4.3.2, that an agent's similarity is reflected in how quickly its IS ratio drops, suggests a simple strategy to adjust for agents that end early due to crashes. To counteract the dependence on trajectory length, we extend their IS ratios using the average rate of the specific trajectory. This is comparable to an additional approach taken in [DTB18], where an estimator is proposed that reweights the trajectories based on their average IS ratio.

However, in our setting we are faced with one additional challenge: our agents exhibiting highly similar behaviors during the initial timesteps, leading to initial IS ratios that are nearly identical and very close to one. If an agent crashes during these early stages, extending the IS ratio with the mean can lead to significant overestimation of the IS ratio for this trajectory in later timesteps.

To address this and improve the performance of our estimator in our specific setting, we introduce a hyperparameter t_{start} that defines a minimum trajectory length and specifies the starting point for applying the mean extension method. For our purpose, we set this to $t_{start} = 15$.

Unfortunately, in addition to requiring an additional hyperparameter, our adjusted Mean-Extended PDWIS estimator is no longer consistent. To see this, simply consider a scenario with a target agent that crashes before t_{start} and utilizes our lifetime reward - even if we have infinite data, our estimator can never converge to the true value.

We will denote this version with PDWIS (Mean).

PDWIS (Zero)

Considering an alternative method, one might think to disregard trajectories terminating before the current timestep or, equivalently, set the timestep-wise IS ratios for these trajectories to zero. However, this strategy makes our PDWIS estimator inconsistent.

To illustrate, consider a scenario where our target agent always crashes immediately at timestep one, terminating the trajectory. The true return, V_{target}, would therefore reflect

the agent's return at that initial timestep. Let us now consider that data is collected with multiple behavior agents, with at least some progressing beyond the first timestep. In that case, as we add more data that progressed beyond the first timestep, our PDWIS estimator V begins to diverge. To see this, let's consider the PDWIS (zero) estimator:

$$V = \sum_{t=1}^{\max_\tau T(\tau)} V_t \tag{4.13}$$

Here we adapt V_t, the inner sum of Eq. 4.12, slightly in-order to account for excluding trajectories that have already crashed:

$$V_t = \sum_{\substack{\tau \in \mathcal{D}_b \\ T(\tau) \geq t}} \gamma^{t-1} r_t^\tau \cdot \frac{p_{1:t}^\tau}{\sum_{\substack{\tau \in \mathcal{D}_b \\ T(\tau) \geq t}} p_{1:t}^\tau} \tag{4.14}$$

V_1 will asymptotically approach V_{target}, since the PDWIS estimator is consistent for the first timestep. Nonetheless, the overall estimate V incorporates contributions from timesteps beyond the first. These additional contributions are not inherently zero, especially in scenarios like ours where every reward is positive. Consequently the remainder of the sum $\sum_{t=2}^{\max_\tau T(\tau)} V_t$ is guaranteed to be larger than 0, if the behavior dataset contains trajectories that do not crash in the first timestep. This adds a non vanishing bias term as the number of our samples approaches infinity.

Therefore, adapting our estimator to exclude trajectories where $T(\tau) < t$ leads to inconsistency. Nonetheless, this approach serves as a foundation for the Termination-aware Per-Decision-Weighted Importance Sampling estimator we introduce in the next section. There, we incorporate the likelihood of termination in earlier timesteps such that, as we argue, consistency is restored to our estimator.

4.3.5 Termination-Aware Per-Decision Weighted Importance Sampling

This section first introduces the novel Termination-aware Per-Decision-Weighted Importance Sampling (TPDWIS) estimator and then gives an argument concerning consistency.

The TPDWIS estimator is given as:

$$V_{TPDWIS}^\pi = \sum_{t=1}^{\max_\tau T(\tau)} S_t \sum_{\substack{\tau \in D_b \\ T(\tau) \geq t}} \gamma^{t-1} r_t^\tau \frac{p_{1:t}^\tau}{\sum_{\substack{\tau \in D_b \\ T(\tau) \geq t}} p_{1:t}^\tau} \tag{4.15}$$

with:

$$S_t = \prod_{t' < t} \left(1 - \frac{\sum_{\substack{\tau \in \mathcal{D}_b \\ T(\tau) \geq t'}} \left[\mathbf{1}_{\{T(\tau) = t'\}} p_{1:t'}^\tau \right]}{\sum_{\substack{\tau \in \mathcal{D}_b \\ T(\tau) \geq t'}} p_{1:t'}^\tau} \right) \tag{4.16}$$

Note that, except for the S_t term this estimator is equivalent to the Zero-extended PDWIS estimator described previously. S_t describes the likelihood of not being in a terminating state at a timestep t.

Intuitively, we calculate the expected value at timestep t by multiplying the expected value under the assumption we have not yet crashed, with the probability of actually not having crashed. Notably, also notice the similarities between the PHWIS estimator, Eq. 4.9, and Eq. 4.15. However, note that we calculate the weight differently, and PHWIS calculates the inner sum over only the current timestep. In the following, we show that our estimator is provably consistent and outperforms PHWIS and all other IS estimators significantly in our empirical evaluation (see Section 5.2.3).

Proof of consistency for TPDWIS

To set the groundwork for arguing the consistency of TPDWIS, we look at the findings in [Tho15], which suggest that under the previously discussed mild assumptions, which hold in our setting, a PDWIS estimator given by:

$$V^{\pi}_{PDWIS} = \sum_{t=1}^{\max_\tau T(\tau)} X_t \tag{4.17}$$

is consistent if, for each timestep t, it holds that $X_t \xrightarrow{a.s.} \mathbb{E}[r_t|\tau \sim \pi]$ [Tho15]. We will show that this holds for every timestep in TPDWIS. Formally, our proof is concluded if we can show:

$$\forall t \leq \max_\tau T(\tau) : S_t \sum_{\substack{\tau \in D_b \\ T(\tau) \geq t}} \gamma^{t-1} r^\tau_t \frac{p^\tau_{1:t}}{\sum_{\substack{\tau \in D_b \\ T(\tau) \geq t}} p^\tau_{1:t}} \xrightarrow{a.s.} \mathbb{E}[r_t|\tau \sim \pi] \tag{4.18}$$

To this aim, we rewrite the expected reward in each timestep with the law of total expectation, considering the termination time $T(\tau)$ of the trajectories:

$$\begin{aligned} \mathbb{E}[r_t|\tau \sim \pi] = P(T(\tau) \geq t|\tau \sim \pi) \cdot \mathbb{E}[r_t|\tau \sim \pi, T(\tau) \geq t] \\ + P(T(\tau) < t|\tau \sim \pi) \cdot \mathbb{E}[r_t|\tau \sim \pi, T(\tau) < t] \end{aligned} \tag{4.19}$$

Here, $\mathbb{E}[r_t|\tau \sim \pi, T(\tau) \geq t]$ is the expected reward assuming the trajectory hasn't terminated by timestep t, and $P(T(\tau) \geq t|\tau \sim \pi)$ is the probability that the trajectory has not crashed at timestep t, we will call this the *survival probability*. The second part of the equation accounts for the expected reward if the trajectory has terminated before t, and its corresponding probability. Since $\mathbb{E}[r_t|\tau \sim \pi, T(\tau) < t] = 0$ — no reward is collected post-crash—the equation simplifies to:

$$\mathbb{E}[r_t|\tau \sim \pi] = P(T(\tau) \geq t|\tau \sim \pi) \cdot \mathbb{E}[r_t|\tau \sim \pi, T(\tau) \geq t] \tag{4.20}$$

Therefore, clearly, we can rewrite the right side of Eq. 4.18 with the right side of Eq. 4.20. We will then show that the following two statements hold:

$$S_t \xrightarrow{a.s.} P(T(\tau) \geq t | \tau \sim \pi) \tag{4.21}$$

$$\sum_{\substack{\tau \in D_b \\ T(\tau) \geq t}} \gamma^{t-1} r_t^\tau \frac{p_{1:t}^\tau}{\sum_{\substack{\tau \in D_b \\ T(\tau) \geq t}} p_{1:t}^\tau} \xrightarrow{a.s.} \mathbb{E}[r_t | \tau \sim \pi, T(\tau) \geq t] \tag{4.22}$$

We remark that the left sides of both Eq. 4.21 and Eq. 4.22 are independent of each other, since each trajectory in each of the different sets considered is generated independently and there is no overlap. Therefore, as per the Continuous Mapping Theorem, we can combine the terms with multiplication on both sides of the $\xrightarrow{a.s.}$ operator. This then yields our consistent TPDWIS estimator as given in Equation 4.15, with the expectation written as on the right side of Eq. 4.20. Hence, it just remains to be seen that Eq. 4.21 and Eq. 4.22 are valid:

The Survival Probability: We will first rewrite the survival probability into the cumulative product of not crashing at any prior timestep t':

$$P(T(\tau) \geq t | \tau \sim \pi) = \prod_{t' < t} \left[1 - P(T(\tau) = t' | T(\tau) \geq t', \tau \sim \pi)\right] \tag{4.23}$$

Note that in each timestep t', we use the likelihood under the condition of not having yet crashed (e.g. the condition $T(\tau) \geq t$) such that the survival probability is calculated properly. We can write $P(T(\tau) = t' | T(\tau) \geq t', \tau \sim \pi)$ as a simple WIS estimate of the likelihood of crashing at timestep t', excluding timestep where we have crashed, which we know to be consistent:

$$\frac{\sum_{\substack{\tau \in \mathcal{D}_b \\ T(\tau) \geq t'}} \left[\mathbf{1}_{\{T(\tau) = t'\}} p_{1:t'}^\tau\right]}{\sum_{\substack{\tau \in \mathcal{D}_b \\ T(\tau) \geq t'}} p_{1:t'}^\tau} \xrightarrow{a.s.} P(T(\tau) = t' | T(\tau) \geq t', \tau \sim \pi) \tag{4.24}$$

Plugging the results from Eq. 4.24 into Eq. 4.23 is valid since the crashing probabilities of subsequent timesteps are independent; doing so gives us our survival probability term as in Eq. 4.16. Furthermore, the consistency of our WIS estimator is not compromised by excluding trajectories, as earlier crashing trajectories cannot have an influence on the crashing likelihood at t'. We can conclude that Eq. 4.21 holds.

The Expectation: Again, as for the survival probability, we argue that the left side of Eq. 4.22 is equivalent to a WIS estimator, which we know to be consistent. Excluding trajectories that have already crashed before t' does not make the estimator inconsistent, as these trajectories cannot have any influence on the estimated reward given that t' was reached.

Finally, as we have shown that both Eq. 4.21 and Eq. 4.22 hold, we have shown that Eq. 4.15 is a consistent estimator as argued earlier.

4.3.6 Evaluation procedure for Importance Sampling

Equation 2.9 reveals that the cumulative IS ratio, and therefore our reward estimation, is conditioned on the initial state $s_1^{\tau_b}$ of our behavior trajectory. Hence, IS will always return an estimate of our reward, given we start in $s_1^{\tau_b}$. To evaluate our estimate, we would need multiple evaluation Monte Carlo rollouts starting exactly in $s_1^{\tau_b}$. In a continuous real-world settings, it's not possible to ensure that all trajectories commence from the same initial state.

Consequently, for each evaluation starting point, we pick all behavioral trajectories that start within a radius r_{start}, which we set to 1 meter, and consider them as belonging to the same starting state for purposes of calculating our IS estimate and evaluation. Note that we do not change the actual starting states in the trajectories, we simply do this to obtain a set of trajectories $\mathcal{D}_b$ that we can provide to our estimator. Subsequently, we compare the results of the estimation with the according evaluation result. This method might introduce bias and make the performance of IS worse when compared with the other estimators, which we may ask to perform estimation from any initial state. However, as this is a limitation of IS in general, and there are no obvious solutions to this issue, we believe our evaluation method to be justified.

4.4 Doubly Robust Estimator

We combine different IS estimators with FQE-DD for our investigation of the DR method. Specifically, we investigate the performance of DR with IS, PDIS, WIS and TPDWIS.

The DR estimator is doubly robust because it retains the benefit of both IS and FQE to improve the estimate. The original formulation of the DR estimator from [JL16] for OPE is recursive and restricted to the finite horizon setting. [TB16] provides a more practical formulation of the estimator that we present here. The formulation in [TB16] transforms our single-step rewards according to the following formula:

$$\hat{r}_t^\tau = r_t^\tau - Q^\pi(s_t^\tau, a_t^\tau) + \gamma V^\pi(s_{t+1}^\tau) \tag{4.25}$$

After transforming the reward according to Equation 4.25 we can utilize any IS-based method together with the transformed rewards $\hat{r}$, estimating $\hat{V}_{*IS}^\pi$. After performing IS, our final estimator is then calculated as follows:

$$\mathrm{V}_{DR}^\pi = \hat{V}_{*IS}^\pi + \frac{1}{|D_b|} \sum_{\forall \tau \in D_b} V^\pi(s_1^\tau) \tag{4.26}$$

We utilize FQE-DD to compute both $Q^\pi(s_t^\tau, a_t^\tau)$ and $V^\pi(s_{t+1}^\tau)$. Computing $Q^\pi(s_t^\tau, a_t^\tau)$ is straightforward since FQE is defined as a Q-function. For computing V^π we similarly employ the Q-function of FQE-DD:

$$V^\pi(s_{t+1}^\tau) = \frac{1}{N} \sum_{i=1}^{N} Q(s_{t+1}^\tau, a) \quad \text{where} \quad a \sim \pi(s_{t+1}^\tau) \tag{4.27}$$

For calculating the adjusted rewards, as given in Eq. 4.25, we estimate $V^{\pi}(s^{\tau}_{t+1})$ with $N = 5$ samples per step-wise reward. For the value function in Eq. 4.26, we sample a total of 20 actions per initial state.

CHAPTER 5

The Real-World F1TENTH Benchmark

In this chapter, we benchmark the performances of the OPE methods investigated within this thesis. To this aim, Section 5.1 explains the overall evaluation protocol employed to compare different OPE methods. Including the evaluation metrics we adopt. Section 5.2, discusses the performance of the different OPE methods when compared within their respective OPE method families. Section 5.3 compares the best-performing methods from each family and analyses differences for different reward signals. Finally, in Section 5.4, we give practical guidelines on selecting OPE methods.

5.1 Evaluation Protocol

We evaluate the performance of an OPE method by supplying the method with our training dataset and all our 15 evaluation agents $\{\pi_1, \pi_2, \dots \pi_{15}\}$. For each of the agents π_i, the OPE method produces a value estimate $\hat{V^{\pi_i}}$, yielding an array of value estimates $\hat{V^{\pi}} = \{\hat{V^{\pi_1}}, \hat{V^{\pi_2}}, \dots \hat{V^{\pi_{15}}}\}$. As previously outlined in Section 3.3, during our data collection procedure, we collected estimates of the ground truth performances of the evaluation agents: $V^{\pi} = \{V^{\pi_1}, V^{\pi_2}, \dots V^{\pi_{15}}\}$. These ground truth estimates are computed over at least 50 trajectories performed with the respective evaluation agent. Comparing estimates $\hat{V^{\pi}}$ with ground truths V^{π}, we compute different evaluation metrics.

As discussed by [FNN+21], evaluation of OPE methods should not be confined to metrics that solely take into account the direct value difference between V^{π_i} and $\hat{V^{\pi_i}}$. Instead, the performance measurement of OPE should also consider the ability of the OPE method to rank the policies and select the best policy correctly, irrespective of the absolute estimated value . Both of these tasks are important in real-world OPE applications, as we might be interested in selecting the best-performing policies from a set of policies and might not be only interested in the concrete predicted raw value estimate.

Hence, we adopt the same metrics as utilized by [FNN+21] in the DOPE benchmark.

Spearman Rank Correlation: This metric measures how well our estimator can rank our different agents, irrespective of the absolute estimated values. The SRC is calculated as:

$$M_{SRC} = \frac{\mathrm{cov}(Ra(V^{\pi}), Ra(\hat{V}^{\pi}))}{\sigma_{Ra(V^{\pi})}\sigma_{Ra(\hat{V}^{\pi})}} \tag{5.1}$$

where $Ra(\cdot)$ is a function that converts our array of value estimates to a ranking. For instance, we might map the highest value in V^{π} to one, the second highest to two, and so on. This step removes the dependence on absolute values, focusing only on the ranking. cov denotes the covariance between the embedded rank variables:

$$\mathrm{cov}(X, Y) = \frac{(X - \mathbb{E}[X]) \cdot (Y - \mathbb{E}[Y])}{|X|} \tag{5.2}$$

Furthermore, σ denotes the standard deviation of our rank variables.

The SRC ranges from -1 to 1. A value of 1 indicates perfect ranking alignment with the ground truths, while -1 signifies an inverse correlation. An SRC of zero implies that the ranking is essentially random, indicating that our OPE method produces no monotonicity concerning the ground truths. In summary, values closer to 1 indicate a better-performing OPE method in terms of ranking. Figure 5.1 shows scatter plot examples of the aforementioned concrete SRC values.

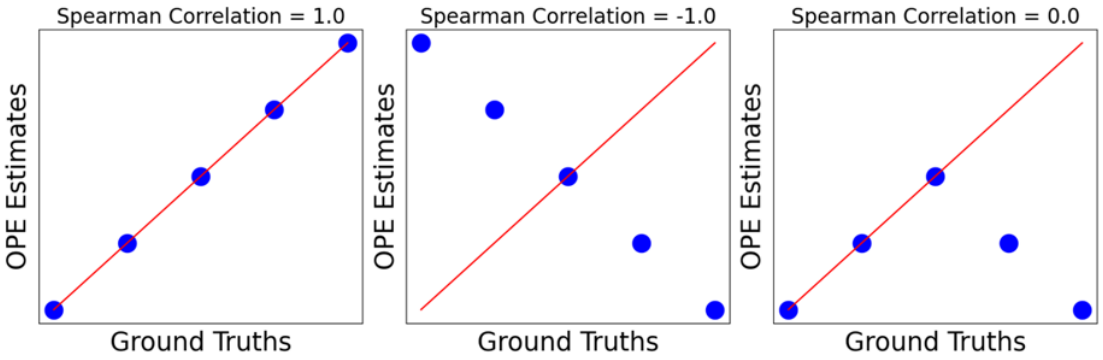


Figure 5.1: Scatter plot depicting multiple agents with varying SRC values. Each blue dot represents a distinct agent π_i, with the Ground Truth Performance on the x-axis and the corresponding OPE estimate on the y-axis. The red line indicates the expected results for a perfect OPE method.

Regret@1: This metric measures the ground truth difference between the best policy according to our estimator and the actual best policy. It is calculated as:

$$M_{R@1} = \left| \max_{\pi_i}(V^{\pi_i}) - V^{\pi_{\text{best}}} \right| \tag{5.3}$$

with

$$\pi_{\text{best}} = \arg\max_{\pi_i}(\hat{V}^{\pi_i}) \tag{5.4}$$

High values indicate that the best policy suggested by our estimator significantly deviates from the actual best policy. Conversely, low values indicate better estimator performance, suggesting closer alignment with the actual best policy. As this metric is only affected by a few good-performing policies, it is much more prone to outliers.

Mean Absolute Error: Contrary to the previous metrics, this metric does not only focus on ranking performance but also the accuracy of the magnitude of the estimate. We calculate the MAE by calculating the mean of absolute per-agent errors. Hence:

$$M_{MAE} = \frac{\sum_{i=1}^{|V^\pi|} |V^{\pi_i} - \hat{V}^{\pi_i}|}{|V^\pi|} \tag{5.5}$$

Consequently, low values indicate better performance. Previous literature commonly employs Mean Squared Error as a performance metric. However, like [FNN+21], we opt for MAE in order to properly assess robustness with respect to outliers.

5.2 Performance of Off-policy Evaluation Methods

We now discuss the performance of our OPE estimators within their respective OPE method families.

5.2.1 Model-based

Table 5.1 presents the results for most of our investigated Model-based methods. These results are visualized in Figures 5.2, 5.3, and 5.4, showing the SRC, MAE, and R@1, respectively. As outlined in Section 4.1, we also investigate model ensembles, with their results presented in Table 5.2.

We discuss the performance of our methods in four distinct groups:

- **Simulation:** As the most commonly used OPE method in real-world robotics applications and its undeniable importance, we discuss the performance of simulation first. Specifically pointing out the shortcomings of the simulator for our environment.
- **Literature:** These methods have been adopted with minimal changes from the literature and include: Naive Model (Mean-Squared-Error) (NM (MSE)), Naive Model (Log-Likelihood) (NM (LL)), Delta Model (Mean-Squared-Error) (DM (MSE)) and the Autoregressive Model (AM).
- **Adaptions:** These methods have been significantly adjusted from the literature. The specific adaptations are discussed in Section 4.1. These methods are Delta Model (Log-Likelihood) (DM (LL)) and Autoregressive Delta Model (ADM). ADM (5) denotes our best-performing ensemble model, consisting of five distinct ADM models. We will discuss only this ensemble in this group.

- **Ensembles:** Finally, we more generally discuss the performance of ensembles of different MB methods.

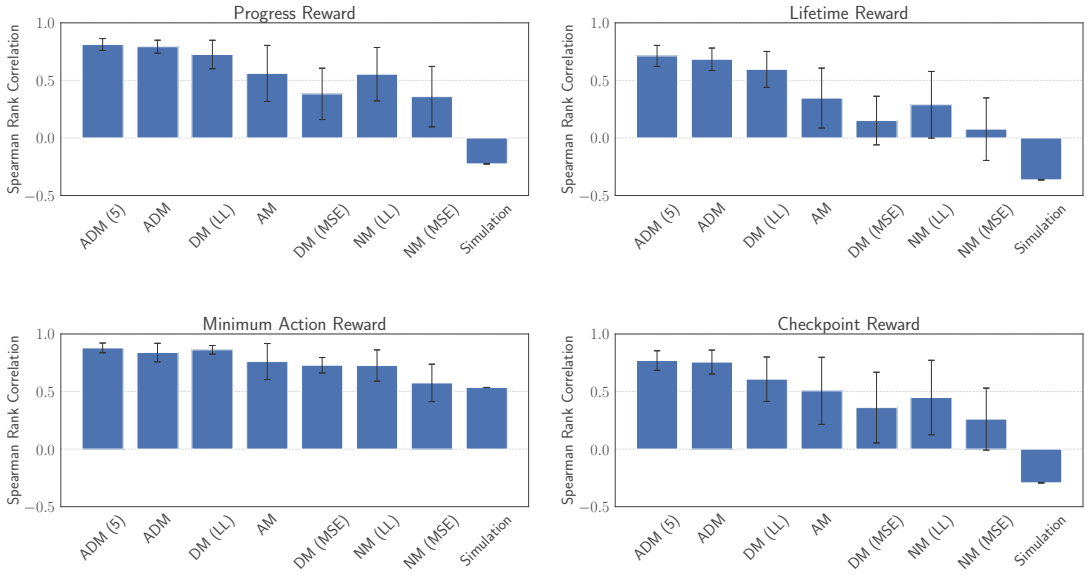


Figure 5.2: SRC for our different MB OPE methods and the rewards investigated.

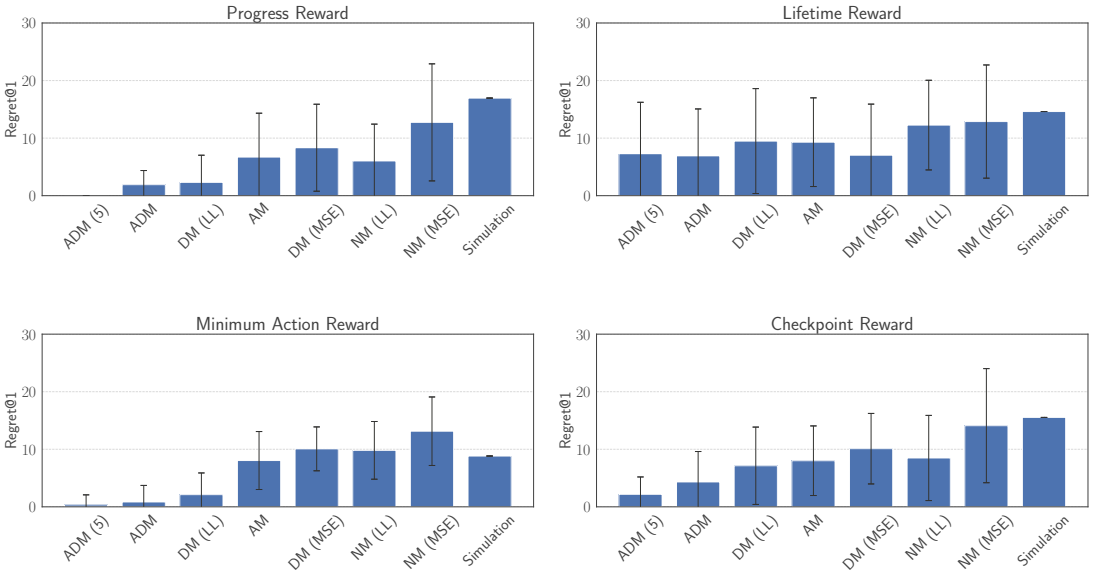


Figure 5.3: R@1 for our different MB OPE methods and the rewards investigated.

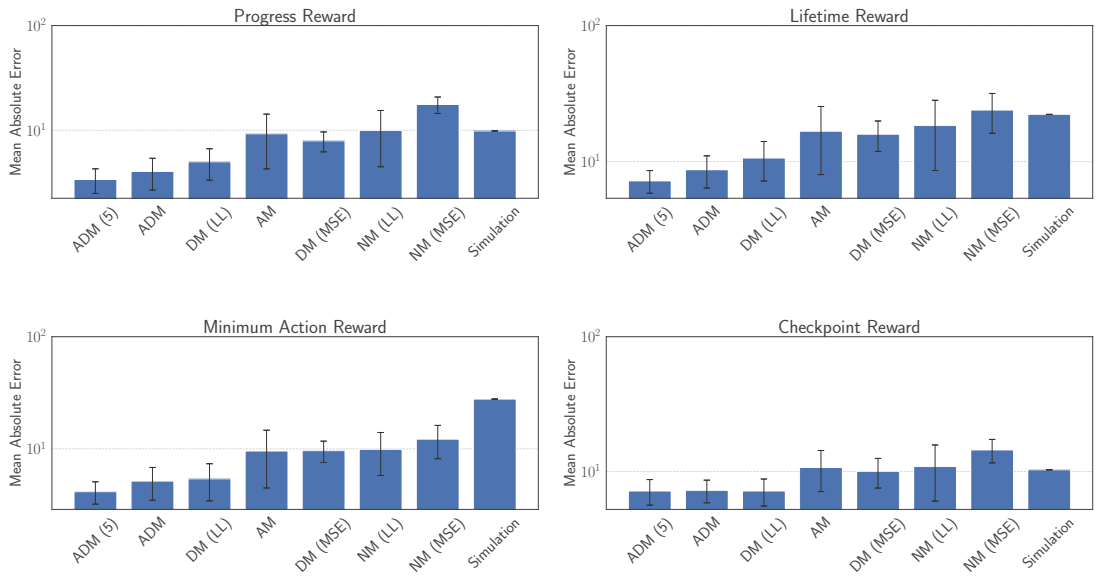


Figure 5.4: MAE for our different MB OPE methods and the rewards investigated.

Simulation

In Table 5.1 and the associated Figures 5.2, 5.3, and 5.4, it can be observed that Simulation is generally the worst performing method or among the worst performing methods across all our metrics and rewards. In this section, we will investigate some of the reasons for this gap.

Figure 5.5 shows scatter plots of simulation estimates versus ground truth performance for our four rewards. We highlight two sets of agents that are significant outliers in the simulation estimates compared to the ground truth: PP agents with very short lookahead distances, highlighted in red, and a subset of fast FTG agents, highlighted in yellow. Furthermore, Figure 5.6 displays real-world and simulation rollouts of two agent configurations from these subsets.

Pure-pursuit with short lookahead: These pure-pursuit agents have very short lookahead distances (< 0.5 meters). In Figure 5.5, we observe that the simulation overestimates the performance of these agents concerning the Lifetime Reward. The same pattern holds for the Minimum Action Reward, where the overestimation is particularly significant. This discrepancy arises because a short lookahead distance induces significant oscillatory behavior in the actual trajectory compared to the racing line in the real world. This oscillatory behavior increases the crash rate in the real world. Furthermore, the Minimum Action Reward is especially impacted, as oscillations are associated with a very low Minimum Action Reward. However, this oscillatory behavior is absent in the simulator, as evidenced by the plotted trajectories in Figure 5.6.

Table 5.1: Benchmark results for MB methods. Best results are highlighted, in case the ensemble model is the best-performing method, second-best results are additionally highlighted.

	Estimator	Spearman (↑)	Mean Absolute Error (↓)	Regret@1 (↓)
Progress Reward	ADM (5)	**0.81 ± 0.05**	**3.39 ± 0.89**	**0.00 ± 0.00**
	ADM	**0.79 ± 0.06**	**4.04 ± 1.35**	**1.91 ± 2.45**
	DM (LL)	0.72 ± 0.12	5.00 ± 1.66	2.31 ± 4.72
	AM	0.56 ± 0.24	9.26 ± 4.99	6.71 ± 7.62
	DM (MSE)	0.38 ± 0.22	7.93 ± 1.71	8.33 ± 7.56
	NM (LL)	0.55 ± 0.23	9.97 ± 5.48	6.04 ± 6.39
	NM (MSE)	0.36 ± 0.26	17.63 ± 3.15	12.74 ± 10.17
	Simulation	-0.22 ± 0.00	9.86 ± 0.00	16.95 ± 0.00
Lifetime Reward	ADM (5)	**0.71 ± 0.09**	**7.22 ± 1.37**	7.29 ± 8.93
	ADM	**0.68 ± 0.10**	**8.72 ± 2.32**	**6.92 ± 8.15**
	DM (LL)	0.60 ± 0.16	10.63 ± 3.42	9.48 ± 9.11
	AM	0.35 ± 0.26	16.72 ± 8.69	9.29 ± 7.70
	DM (MSE)	0.15 ± 0.21	15.89 ± 4.00	7.03 ± 8.88
	NM (LL)	0.29 ± 0.29	18.41 ± 9.81	12.26 ± 7.79
	NM (MSE)	0.08 ± 0.27	23.93 ± 7.77	12.88 ± 9.83
	Simulation	-0.36 ± 0.00	22.26 ± 0.00	14.62 ± 0.00
Min Action Reward	ADM (5)	**0.88 ± 0.04**	**4.15 ± 0.94**	**0.35 ± 1.73**
	ADM	0.84 ± 0.08	**5.15 ± 1.67**	**0.85 ± 2.88**
	DM (LL)	**0.86 ± 0.04**	5.40 ± 1.96	2.12 ± 3.77
	AM	0.76 ± 0.16	9.56 ± 5.09	8.04 ± 5.03
	DM (MSE)	0.73 ± 0.07	9.62 ± 2.08	10.07 ± 3.81
	NM (LL)	0.73 ± 0.14	9.89 ± 4.10	9.81 ± 5.01
	NM (MSE)	0.57 ± 0.16	12.18 ± 4.02	13.14 ± 5.95
	Simulation	0.54 ± 0.00	27.77 ± 0.00	8.83 ± 0.00
Checkpoint Reward	ADM (5)	**0.77 ± 0.08**	**7.17 ± 1.54**	**2.18 ± 3.02**
	ADM	**0.76 ± 0.10**	7.25 ± 1.38	**4.33 ± 5.28**
	DM (LL)	0.61 ± 0.19	**7.17 ± 1.63**	7.14 ± 6.71
	AM	0.51 ± 0.29	10.71 ± 3.61	8.01 ± 6.05
	DM (MSE)	0.36 ± 0.31	10.03 ± 2.49	10.11 ± 6.13
	NM (LL)	0.45 ± 0.32	10.89 ± 4.86	8.49 ± 7.41
	NM (MSE)	0.26 ± 0.27	14.46 ± 2.87	14.10 ± 9.90
	Simulation	-0.29 ± 0.00	10.30 ± 0.00	15.54 ± 0.00

Fast FTG: As observed in Figure 5.5, the performance of FTG agents configured to be particularly fast is significantly underestimated in simulation (the yellow agents in the Figure). These agents crash much sooner in simulation than in reality, affecting

Table 5.2: Evaluation of ensemble estimators for different rewards.

	Estimator	Spearman (↑)	Mean Absolute Error (↓)	Regret@1 (↓)
Progress Reward	ADM	0.79 ± 0.06	4.04 ± 1.35	1.91 ± 2.45
	ADM (5)	**0.81 ± 0.05**	**3.39 ± 0.89**	**0.00 ± 0.00**
	AM	0.56 ± 0.24	9.26 ± 4.99	6.71 ± 7.62
	AM (5)	**0.68 ± 0.17**	**5.32 ± 2.15**	**3.53 ± 5.29**
	DM (LL)	0.72 ± 0.12	5.00 ± 1.66	2.31 ± 4.72
	DM (LL) (5)	**0.75 ± 0.12**	**4.31 ± 1.11**	**0.70 ± 2.14**
	NM (MSE)	0.36 ± 0.26	17.63 ± 3.15	12.74 ± 10.17
	NM (MSE) (5)	**0.46 ± 0.23**	**13.80 ± 2.93**	**8.85 ± 8.57**
Lifetime Reward	ADM	0.68 ± 0.10	8.72 ± 2.32	**6.92 ± 8.15**
	ADM (5)	**0.71 ± 0.09**	**7.22 ± 1.37**	7.29 ± 8.93
	AM	0.35 ± 0.26	16.72 ± 8.69	**9.29 ± 7.70**
	AM (5)	**0.41 ± 0.24**	**10.93 ± 3.80**	11.06 ± 8.41
	DM (LL)	0.60 ± 0.16	10.63 ± 3.42	**9.48 ± 9.11**
	DM (LL) (5)	**0.62 ± 0.11**	**9.07 ± 1.95**	10.33 ± 8.67
	NM (MSE)	**0.08 ± 0.27**	23.93 ± 7.77	**12.88 ± 9.83**
	NM (MSE) (5)	0.03 ± 0.25	**17.10 ± 8.09**	14.90 ± 8.88
Min Action Reward	ADM	0.84 ± 0.08	5.15 ± 1.67	0.85 ± 2.88
	ADM (5)	**0.88 ± 0.04**	**4.15 ± 0.94**	**0.35 ± 1.73**
	AM	0.76 ± 0.16	9.56 ± 5.09	8.04 ± 5.03
	AM (5)	**0.82 ± 0.09**	**6.22 ± 2.52**	**5.48 ± 5.30**
	DM (LL)	0.86 ± 0.04	5.40 ± 1.96	2.12 ± 3.77
	DM (LL) (5)	**0.87 ± 0.05**	**4.85 ± 1.38**	**0.00 ± 0.00**
	NM (MSE)	0.57 ± 0.16	12.18 ± 4.02	**13.14 ± 5.95**
	NM (MSE) (5)	**0.62 ± 0.13**	**11.22 ± 2.72**	13.57 ± 5.32
Checkpoint Reward	ADM	0.76 ± 0.10	7.25 ± 1.38	4.33 ± 5.28
	ADM (5)	**0.77 ± 0.08**	**7.17 ± 1.54**	**2.18 ± 3.02**
	AM	0.51 ± 0.29	10.71 ± 3.61	8.01 ± 6.05
	AM (5)	**0.58 ± 0.25**	**9.03 ± 2.84**	**7.86 ± 6.35**
	DM (LL)	0.61 ± 0.19	7.17 ± 1.63	7.14 ± 6.71
	DM (LL) (5)	**0.65 ± 0.14**	**7.10 ± 1.65**	**4.82 ± 6.15**
	NM (MSE)	0.26 ± 0.27	14.46 ± 2.87	14.10 ± 9.90
	NM (MSE) (5)	**0.30 ± 0.32**	**10.38 ± 3.25**	**11.35 ± 8.34**

all our rewards, as they are set to zero for timesteps after a crash. The exact reasons for this simulation-reality mismatch are harder to pinpoint compared to the previous set of agents. However, investigating Figure 5.6 reveals that, in reality, the agent drifts much more to the outside of curves. This discrepancy might be due to parameters in the

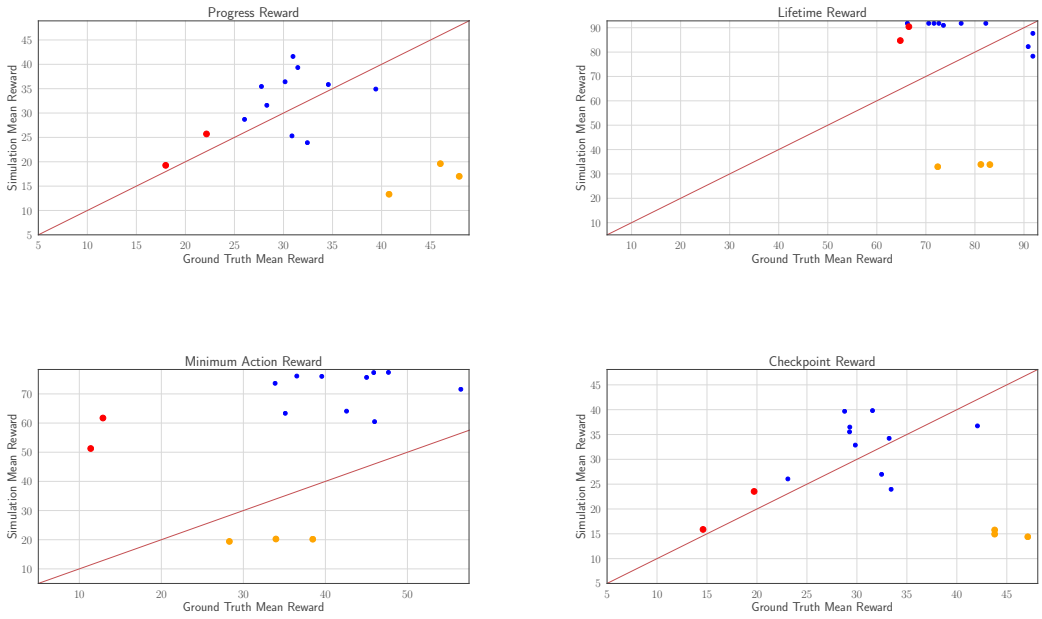


Figure 5.5: Scatter plot of simulation estimates vs. ground truth performance for our evaluation agents. Each dot represents one agent. Red dots indicate PP agents with a short lookahead distance. Yellow dots signify a subset of very fast FTG agents.

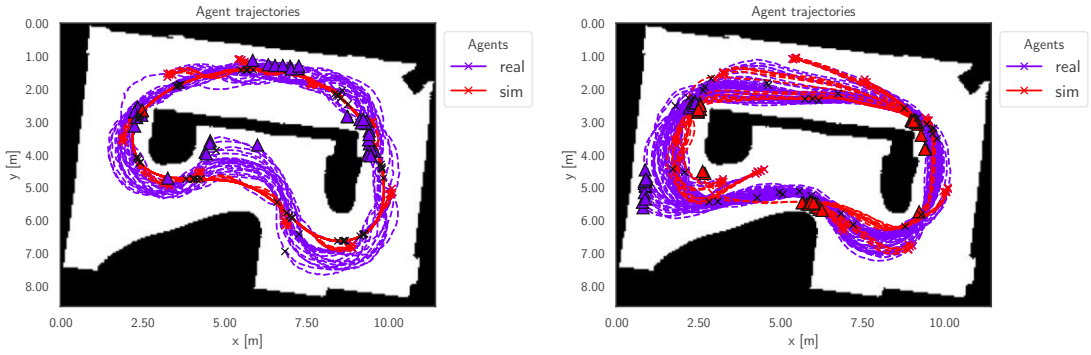


Figure 5.6: Both figures show a comparison of the recorded real-world rollouts versus their simulated rollouts, starting from the same initial state. The left figure depicts a PP agent with a short lookahead distance. The right figure shows a fast FTG agent.

simulator, such as the friction coefficient, being misaligned with reality.

Methods from Literature

In this group of methods, the Autoregressive Model consistently outperforms all the other methods across our rewards and benchmarking metrics. Furthermore, regarding the naive model, the LL loss consistently outperforms the MSE loss. It is worth noting that this loss is already employed in the AM model, and the MSE loss was not investigated in relation to this model. Lastly, concerning R@1 and SRC, there appears to be no clear trend as to whether DM (MSE) outperforms NM (MSE), with DM (MSE) performing better in roughly half of the metric and reward combinations. However, in terms of MAE, DM (MSE) clearly outperforms NM (MSE) across every reward.

Adaptions from Literature

Following the previous results, we combined the LL loss and the DM, as described in Section 4.1. This combination led to the most significant improvement in the Lifetime Reward, with a more than 100% increase in the SRC metric compared to the previous best AM method from the literature.

Additionally, we combined the AM and the DM, yielding ADM, as outlined in Section 4.1. This combination resulted in further improvements across almost all our rewards and metrics. In cases where ADM is not the best-performing method, it is very close, as shown in Table 5.1. We conclude that from our adaptions ADM performs the best.

Performance of Ensembles

As can be observed in Table 5.2, utilizing ensembles as described in Section 4.1.5 generally leads to better-performing models, except for the naive model. Furthermore, this trend is not always visible in the R@1 metric. However, as this metric is more sensitive to changes in performance estimation of only a few agents, we conclude that employing ensembles tends to improve performance.

5.2.2 Fitted-Q-Evaluation

For FQE, we investigate two methods from literature and report results in Table 5.3. Concerning the SRC and MAE metrics, FQE-DD tends to outperform FQE-L2, except for the Minimum Action Reward, where both estimators achieve comparable performance. However, in terms of the R@1 metric, FQE-L2 outperforms FQE-DD.

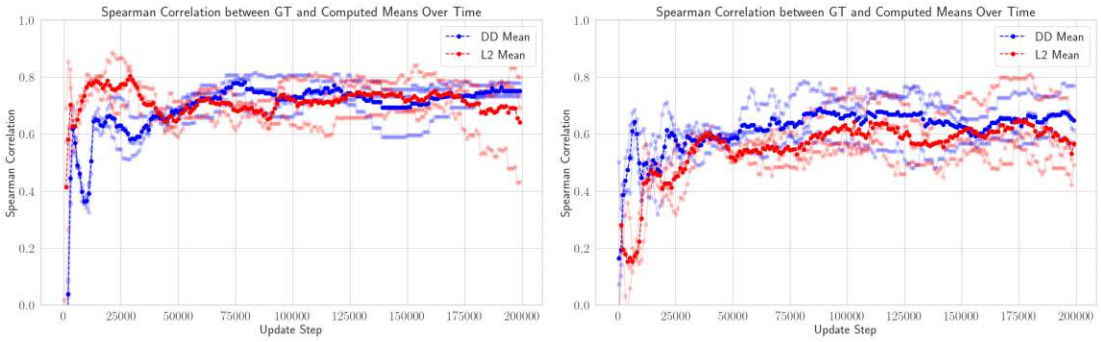
While FQE exhibits the need for hyper-parameter tuning, we do not investigate this extensively due to the comparably large compute resources required to train FQE methods, making extensive tuning infeasible within the scope of this thesis. Furthermore, as the performance of FQE is competitive with our selected hyperparameters and hyper-parameter tuning might be infeasible for real-world applications, we believe that our results are representative of the method's overall potential. For our training setup, on an NVIDIA GeForce RTX 3090, training one FQE model for a single reward and a single agent for 200k steps takes 2-4 hours, depending on the model.

Inline with previous work [PPM+20], since it is not clear when an FQE model has converged, we provide the SRC value over the training steps in Figure 5.7. It should be noted that the SRC reported in the figure is only computed over single action samples, as opposed to the values reported in Table 5.3, explaining the slight inconsistencies between the table and the figures.

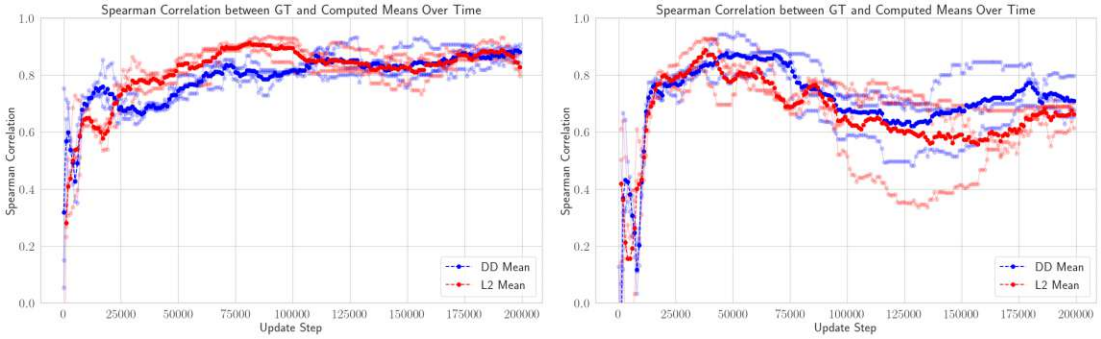
We can observe that, generally, the predicted SRC remains relatively stable over the training steps for our dense rewards. The different training seeds also produce similar results for these rewards across the training steps. On the other hand, our sparser and longer time-horizon rewards result in much higher variance concerning different seeds and appear to exhibit worse convergence. Especially the Checkpoint Reward appears to exhibit significant variance along the training steps. We hypothesise that this is due the methods initially learning to focus on the speed at which the agents drive, only later starting to fit to the sparsity present due to early terminations. However, this requires further investigation.

Table 5.3: Evaluation of estimators for different rewards.

	Estimator	Spearman (↑)	Mean Absolute Error (↓)	Regret@1 (↓)
Progress Reward	FQE DD	**0.75 ± 0.02**	**3.87 ± 0.46**	1.92 ± 0.00
	FQE (L2)	0.70 ± 0.11	4.43 ± 0.29	**1.28 ± 0.91**
Lifetime Reward	FQE (DD)	**0.67 ± 0.05**	**7.04 ± 0.61**	10.27 ± 8.68
	FQE (L2)	0.60 ± 0.10	7.23 ± 1.26	**3.20 ± 4.52**
Min. Action Reward	FQE (DD)	0.86 ± 0.02	**5.57 ± 0.12**	**10.62 ± 0.00**
	FQE (L2)	**0.88 ± 0.02**	6.25 ± 0.26	**10.62 ± 0.00**
Checkpoint Reward	FQE (DD)	**0.73 ± 0.05**	**4.53 ± 0.24**	3.31 ± 0.00
	FQE (L2)	0.65 ± 0.05	5.46 ± 0.23	**2.21 ± 1.56**



(a) SRC as predicted by FQE for the Progress Reward.



(b) SRC as predicted by FQE for the Lifetime Reward.

(c) SRC as predicted by FQE for the Minimum Action Reward.

(d) SRC as predicted by FQE for the Checkpoint Reward.

Figure 5.7: SRC for FQE-DD (blue) and FQE-L2 (red) over the training timesteps. The averaged result is plotted with solid lines, while individual runs are shown with transparency.

5.2.3 Importance Sampling

In Table 5.4, we report the results for our estimators belonging to the IS family. These results are also visualized in Figures 5.8, 5.9, and 5.10. We do not provide standard deviations for IS methods, as these methods are deterministic and only depend on the training data. While we do not include this in our work, it is possible to calculate standard deviations using bootstrapping.

We will proceed with discussing the estimator's performance in two groups:

- **Previous Literature:** These estimators encompass IS, PDIS, WIS, PHWIS, and PDWIS (C). We will also discuss the performance of PDWIS (Mean) in this category, although it could be argued that this estimator has some novel aspects.

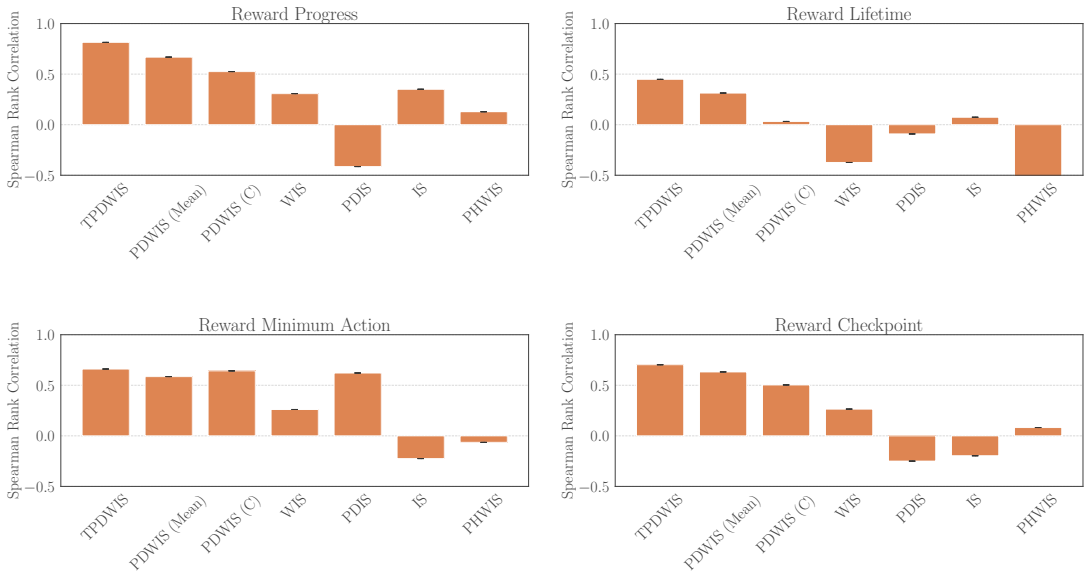


Figure 5.8: SRC for our different IS methods and the rewards investigated.

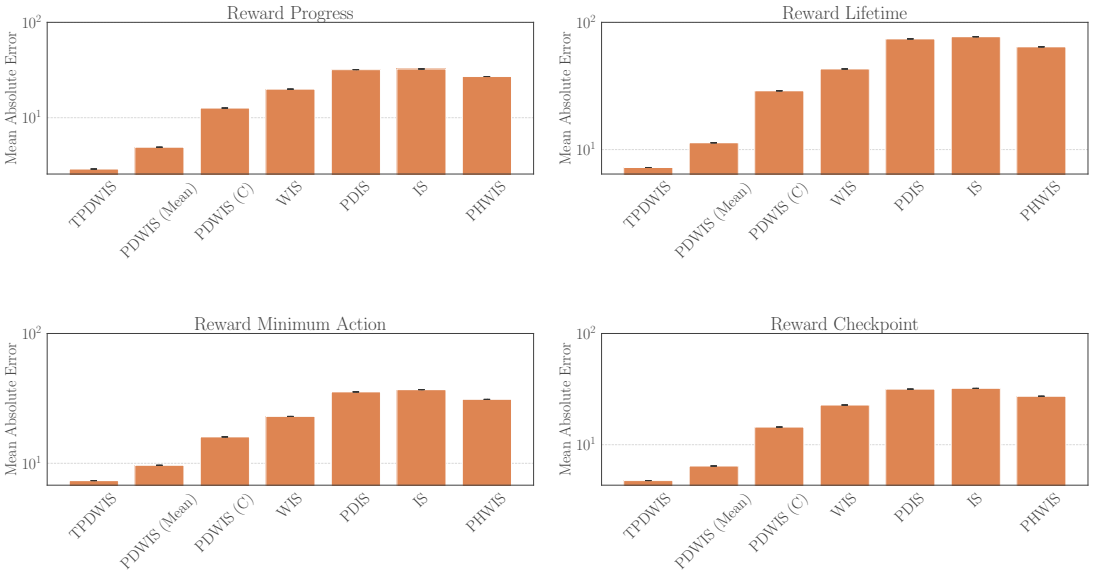


Figure 5.9: MAE for our different IS methods and the rewards investigated.

- **Termination-aware Per-Decision-Weighted Importance Sampling:** Here we will discuss the performance of the novel TPDWIS estimator.

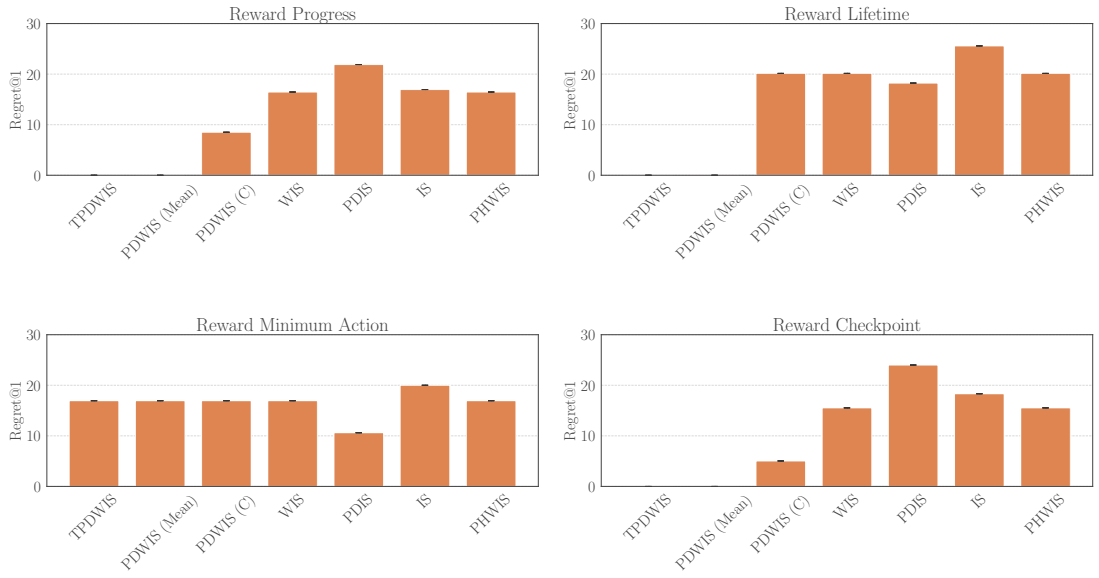


Figure 5.10: R@1 for our different IS methods and the rewards investigated.

Literature

The IS and PDIS estimators perform poorly across most metrics and rewards. The only exception is the Minimum Action Reward, where PDIS performs better than all other IS estimators on R@1. Additionally, the SRC for this reward is also close to the best-performing IS method.

However, the MAE is very high for both these estimators. This is due to the severe underestimation caused by using the unscaled IS ratio, which becomes very close to zero after only a few timesteps in our environment.

While WIS improves performance concerning the MAE metric, the SRC remains below 0.31 for all our rewards. Generally, it does not improve on the previous results from IS and PDIS concerning this metric. In fact, the rank correlation is often negative. The poor performance appears to be due to the early terminations discussed in Section 4.3.2 and present in our training dataset. Since the early terminations are not handled explicitly, the WIS estimator also exhibits severe underestimation of rewards.

The PHWIS estimator was initially designed to address the issue of early terminations. However, our benchmark shows that, for our environment, the performance of PHWIS is often comparable to IS and PDIS, and it is consistently outperformed by plain WIS across the SRC and MAE metrics.

Additionally, we investigated the heuristic PHWIS estimator. However, in our environment, this estimator yields predictions that are often infinite values in floating point 32. Examining the weight calculation for the heuristic PHWIS estimator, as shown in

Table 5.4: Evaluation of Estimators for different Rewards

	Estimator	Spearman (↑)	Mean Absolute Error (↓)	Regret@1 (↓)
Progress Reward	TPDWIS	**0.81 ± 0.00**	**2.89 ± 0.00**	**0.00 ± 0.00**
	PDWIS (Mean)	0.67 ± 0.00	4.90 ± 0.00	**0.00 ± 0.00**
	PDWIS (C)	0.52 ± 0.00	12.62 ± 0.00	8.51 ± 0.00
	WIS	0.31 ± 0.00	19.92 ± 0.00	16.46 ± 0.00
	PDIS	-0.41 ± 0.00	31.90 ± 0.00	21.89 ± 0.00
	IS	0.35 ± 0.00	32.45 ± 0.00	16.95 ± 0.00
	PHWIS	0.13 ± 0.00	27.02 ± 0.00	16.46 ± 0.00
Lifetime Reward	TPDWIS	**0.45 ± 0.00**	**7.23 ± 0.00**	**0.00 ± 0.00**
	PDWIS (Mean)	0.31 ± 0.00	11.31 ± 0.00	**0.00 ± 0.00**
	PDWIS (C)	0.03 ± 0.00	28.97 ± 0.00	20.16 ± 0.00
	WIS	-0.37 ± 0.00	43.02 ± 0.00	20.16 ± 0.00
	PDIS	-0.09 ± 0.00	74.12 ± 0.00	18.23 ± 0.00
	IS	0.07 ± 0.00	77.10 ± 0.00	25.58 ± 0.00
	PHWIS	-0.54 ± 0.00	64.19 ± 0.00	20.16 ± 0.00
Min Action Reward	TPDWIS	**0.66 ± 0.00**	**7.35 ± 0.00**	16.95 ± 0.00
	PDWIS (Mean)	0.59 ± 0.00	9.65 ± 0.00	16.95 ± 0.00
	PDWIS (C)	0.64 ± 0.00	15.98 ± 0.00	16.95 ± 0.00
	WIS	0.26 ± 0.00	22.99 ± 0.00	16.95 ± 0.00
	PDIS	0.62 ± 0.00	35.50 ± 0.00	**10.62 ± 0.00**
	IS	-0.22 ± 0.00	36.91 ± 0.00	20.00 ± 0.00
	PHWIS	-0.06 ± 0.00	31.10 ± 0.00	16.95 ± 0.00
Checkpoint Reward	TPDWIS	**0.70 ± 0.00**	**4.76 ± 0.00**	**0.00 ± 0.00**
	PDWIS (Mean)	0.63 ± 0.00	6.43 ± 0.00	**0.00 ± 0.00**
	PDWIS (C)	0.50 ± 0.00	14.41 ± 0.00	5.04 ± 0.00
	WIS	0.26 ± 0.00	22.74 ± 0.00	15.54 ± 0.00
	PDIS	-0.25 ± 0.00	31.60 ± 0.00	24.01 ± 0.00
	IS	-0.20 ± 0.00	32.13 ± 0.00	18.33 ± 0.00
	PHWIS	0.08 ± 0.00	27.26 ± 0.00	15.54 ± 0.00

Equation 4.9, we notice that the numerator's value of the weight will generally be very high, given that a significant number of our trajectories are truncated at 250 timesteps. Raising these values to the power of $\frac{1}{250}$ results in a numerator that is much larger than our very small denominator, leading to significant overestimations of the true value. To illustrate, let us consider a concrete example: our dataset contains one trajectory that has a cumulative IS ratio of 10^{-100}, a value that is higher (and therefore easier for PHWIS) than realistic values in our setting, as can also be observed in Figure 4.1. Plugging this value into the heuristic PHWIS:

$$W_t^h = \frac{0.99}{10^{-100}} \approx 10^{99} \tag{5.6}$$

If we now consider the rest of the expression in Equation 4.9, for the case of only a single trajectory the remainder of the equation will equal only the product of our discounts till timestep 250, hence $0.99^{250} = 0.08$, a value on a completely different scale than our weight. Even if we have multiple trajectories, the value of the rest of the equation will be completely dominated by the weight, yielding an unusable estimator for our environment. Hence, the results for this estimator are not reported in this benchmark.

Lastly, let us consider the PDWIS estimator. This estimator, even in its Constant extension form previously encountered in the literature, generally outperforms or produces similar results to the best other methods in the literature. The improvement is especially significant for the SRC metric and the MAE. For R@1, an improvement is only present for the Progress and Checkpoint Reward.

We can further improve the performance of PDWIS by extending it with the mean, as described in Section 4.3.4. This consistently improves performance across almost all our metrics and rewards, sometimes significantly. The only major exception is the Minimum Action Reward, where both the SRC and R@1 are not improved but remain similar or equal to the PDWIS (C) estimator.

Termination-aware Per-Decision-Weighted Importance Sampling

The novel TPDWIS estimator significantly outperforms all other IS methods investigated. The previously best-performing estimator from the literature can be considered the PDWIS (C) estimator, as it is debatable whether the mean extension method has been previously utilized. TPDWIS is the **only** estimator that yields a SRC significantly above 0 for the Lifetime Reward when considering only methods from previous literature. Furthermore, the performance concerning SRC is improved by 55%, 3 %, and 40% for Progress, Minimum Action, and Checkpoint Reward, respectively.

Similar significant improvements can be observed for the MAE metric. Regarding R@1, the performance improvement is less clear, but the results are still among the best-performing IS estimators, with only small differences, as in the case of the Checkpoint Reward. The only instance where the TPDWIS estimator produces a notably worse result than other estimators is the Minimum Action Reward, as measured by the R@1 metric. As this metric can be heavily skewed by a few outliers, and the TPDWIS estimator exhibits significant improvements across all other rewards and metrics, we conclude that the TPDWIS estimator is the generally best-performing IS estimator investigated.

5.2.4 Doubly Robust

The results for the investigated DR methods are reported in Table 5.5. Figures 5.11, 5.12 and 5.13 visualize these results and show the performance of FQE-DD and the respective IS methods for the different rewards.

Generally, for IS and PDWIS, as the IS ratios for these methods are vanishingly small, we recover the FQE-DD estimate for the DR method. For WIS, we observe that utilizing DR instead of plain FQE-DD generally decreases the performance of the estimator. However, considering that the original WIS estimator exhibits significantly worse behavior than FQE-DD, it is perhaps surprising how slight the performance degradation is in some cases, like the Lifetime Reward and the Minimum Action Reward. While this requires more investigation, we hypothesize that this is because FQE-DD produces similar value estimates for trajectories that depend only slightly on the concrete starting states.

For the DR version of TPDWIS, we observe similar performance degradation for the Lifetime and Minimum Action Rewards. However, interestingly, for the Checkpoint Reward, we see a significant improvement in both SRC and MAE for the DR estimator, compared to both the FQE-DD and TPDWIS estimators separately. This result should be taken cautiously, as the standard deviation for this estimator is quite large and was calculated with only three available seeds from FQE-DD.

In general, except for the Checkpoint Reward, DR performs slightly worse than FQE-DD alone. However, in many situations, utilizing DR instead of plain FQE does not significantly decreases performance, even if the IS method performs much worse.

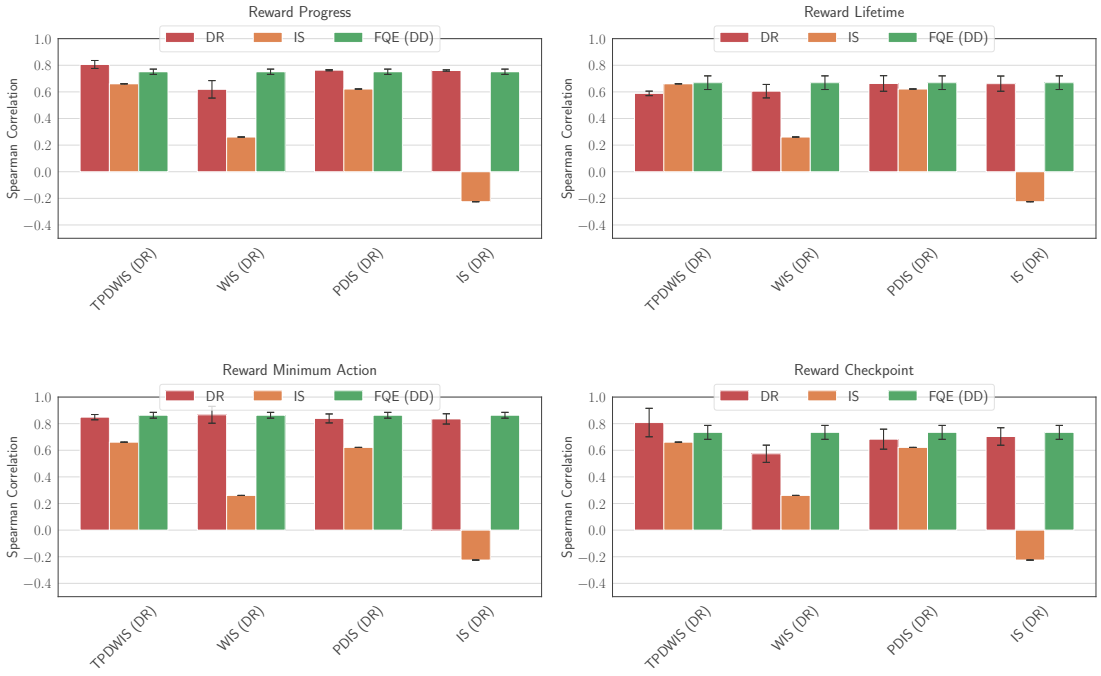


Figure 5.11: SRC for different DR methods, including the performance of the FQE and IS methods they are based on.

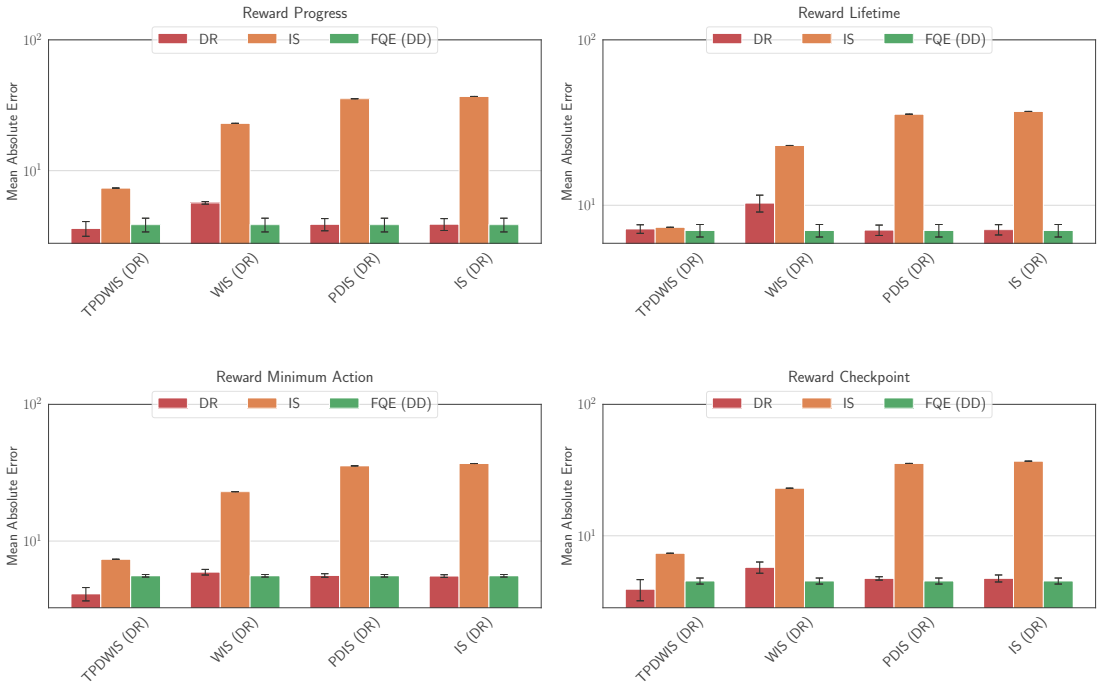


Figure 5.12: MAE for different DR methods, including the performance of the FQE and IS methods they are based on.

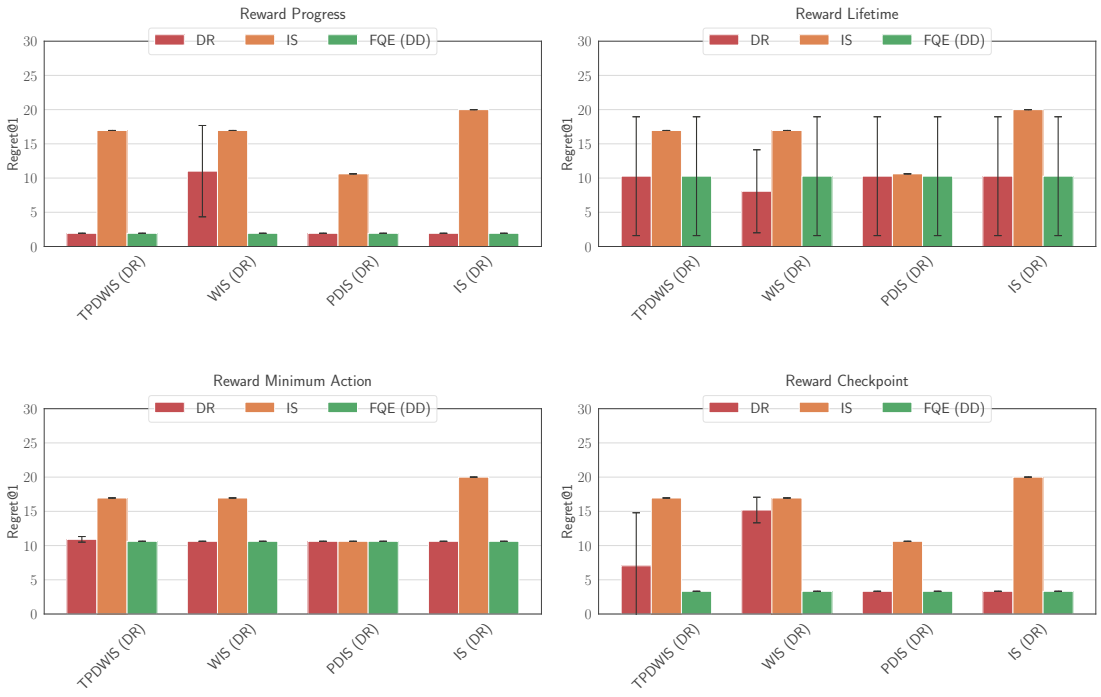


Figure 5.13: R@1 for different DR methods, including the performance of the FQE and IS methods they are based on.

Table 5.5: Evaluation of Estimators for different rewards.

	Estimator	Spearman (↑)	Mean Absolute Error (↓)	Regret@1 (↓)
Progress Reward	TPDWIS (DR)	**0.81 ± 0.03**	**3.62 ± 0.47**	**1.92 ± 0.00**
	WIS (DR)	0.62 ± 0.07	5.67 ± 0.13	11.01 ± 6.67
	PDIS (DR)	0.76 ± 0.00	3.89 ± 0.42	**1.92 ± 0.00**
	IS (DR)	0.76 ± 0.01	3.89 ± 0.41	**1.92 ± 0.00**
Lifetime Reward	TPDWIS (DR)	0.59 ± 0.02	7.19 ± 0.43	10.27 ± 8.68
	WIS (DR)	0.61 ± 0.05	10.31 ± 1.21	**8.07 ± 6.06**
	PDIS (DR)	**0.66 ± 0.06**	**7.08 ± 0.51**	10.27 ± 8.68
	IS (DR)	**0.66 ± 0.06**	7.12 ± 0.51	10.27 ± 8.68
Min. Action Reward	TPDWIS (DR)	0.85 ± 0.02	**4.11 ± 0.46**	10.91 ± 0.41
	WIS (DR)	**0.87 ± 0.06**	5.92 ± 0.28	**10.62 ± 0.00**
	PDIS (DR)	0.84 ± 0.03	5.60 ± 0.17	**10.62 ± 0.00**
	IS (DR)	0.84 ± 0.04	5.55 ± 0.12	**10.62 ± 0.00**
Checkpoint Reward	TPDWIS (DR)	**0.81 ± 0.11**	**3.92 ± 0.72**	7.05 ± 7.75
	WIS (DR)	0.57 ± 0.06	5.75 ± 0.56	15.19 ± 1.87
	PDIS (DR)	0.68 ± 0.08	4.74 ± 0.14	**3.31 ± 0.00**
	IS (DR)	0.70 ± 0.07	4.74 ± 0.29	**3.31 ± 0.00**

5.3 Cross-Method Comparison and Influence of Reward

Figures 5.14, 5.15, and 5.16 show the performance of the methods that perform best within their OPE family. For MB, we show both the performance of ADM (5), as it is clearly the best-performing method, and DM (LL) (5), as it is close on some metrics. For FQE, we show the performance of both versions. For IS, we show the performance of TPDWIS as it is the best-performing IS method, with only a few outliers outperforming it in the more variable R@1 metric. For DR, we elect to show the performance of the respective TPDWIS and PDIS, as these two DR methods show the best performance, except for a single reward and metric: the Minimum Action Reward, where their performance is very close to the best-performing WIS (DR). By selecting these methods we believe that a fair comparison between the best performing methods across rewards and different metrics can be made.

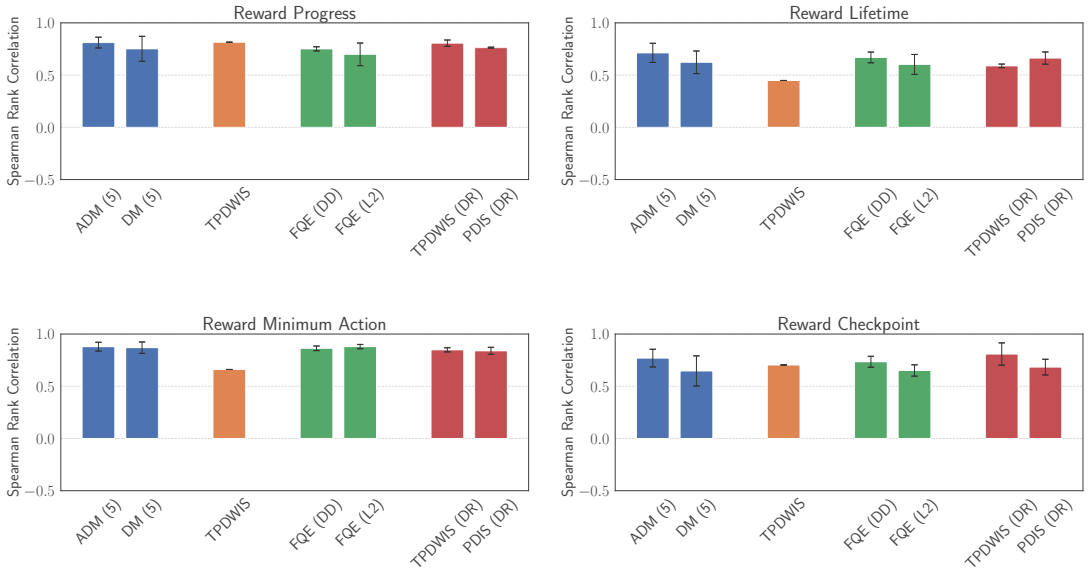


Figure 5.14: SRC for our different OPE methods and the rewards investigated

Progress Reward: The best methods from all OPE families perform very well on the Progress Reward, with similarly high results around ~ 0.8 SRC, < 4.0 MAE, and < 2.0 R@1. Across these metrics, FQE appears to perform the worst, albeit with only a small margin. It is noteworthy that the best methods from previous literature, namely PDWIS (C) for IS and AM for MB, attain significantly worse results than the methods presented in the figures of this section.

Lifetime Reward: Concerning the SRC, the Lifetime Reward appears to be the most challenging reward for our OPE methods to solve, this is perhaps unsurprising as its both sparse and considers a long-time horizon. The best-performing method ADM (5), only achieves a mean SRC of 0.71. FQE and DR achieve similar mean SRC values of 0.67

and 0.66, respectively, lagging only slightly behind our MB method. However, TPDWIS significantly lags behind these results with a notably lower SRC of 0.45. One potential reason for this significantly lower performance might be the inability of IS to perform estimation from any initial state and requiring the evaluation approach described in Section 4.3.6.

On the other hand, all methods perform very comparably on the MAE metric, and TPDWIS significantly outperforms all other methods on the R@1 metric. To summarize, considering that the R@1 metric is significantly influenced by outliers, ADM (5) appears to be the best-performing method for this reward.

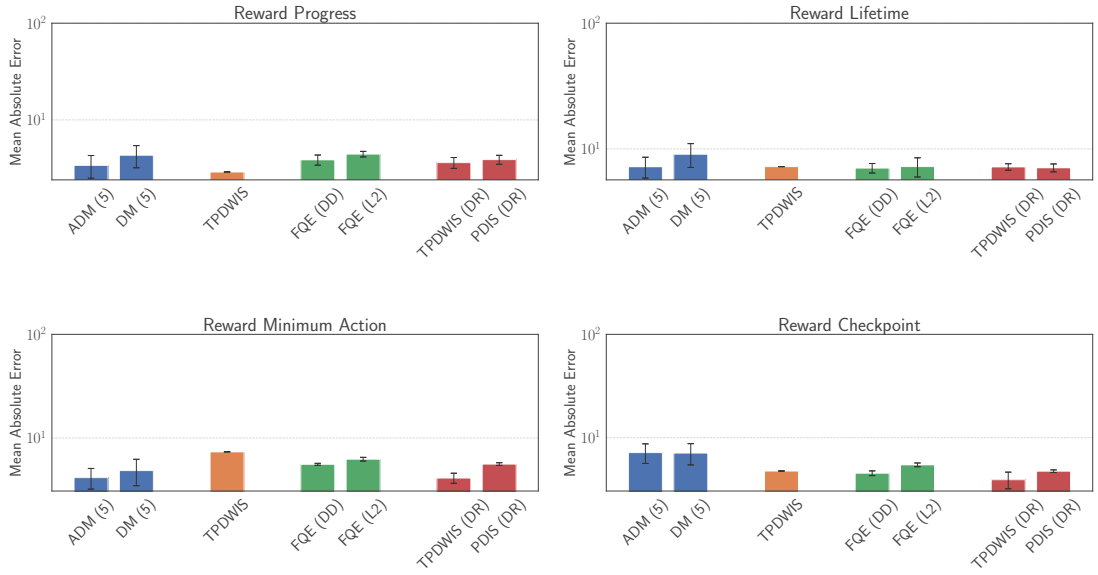


Figure 5.15: MAE for our different OPE methods and the rewards investigated

Minimum Action Reward: The performances of the best FQE, MB, and DR methods are very similar on this reward. Concerning the SRC, FQE-L2 slightly outperforms the other methods, but ADM (5) exhibits the best mean scores in R@1 and Mean Squared Error.

The only method that performs significantly worse than the others is TPDWIS, exhibiting significantly worse SRC, the worst scores on MAE and R@1.

Checkpoint Reward: Concerning the SRC, ADM (5) and TPDWIS (DR) perform best on this metric with scores of 0.77 and 0.81, respectively. While TPDWIS and FQE only attain scores of 0.7 and 0.73. On MAE TPDWIS and TPDWIS (DR) perform best, while on R@1 TPDWIS and ADM (5) perform best. The overall performance of the methods is very similar concerning this reward.

Overall, the dense Progress Reward is handled well by all our OPE methods. In contrast,

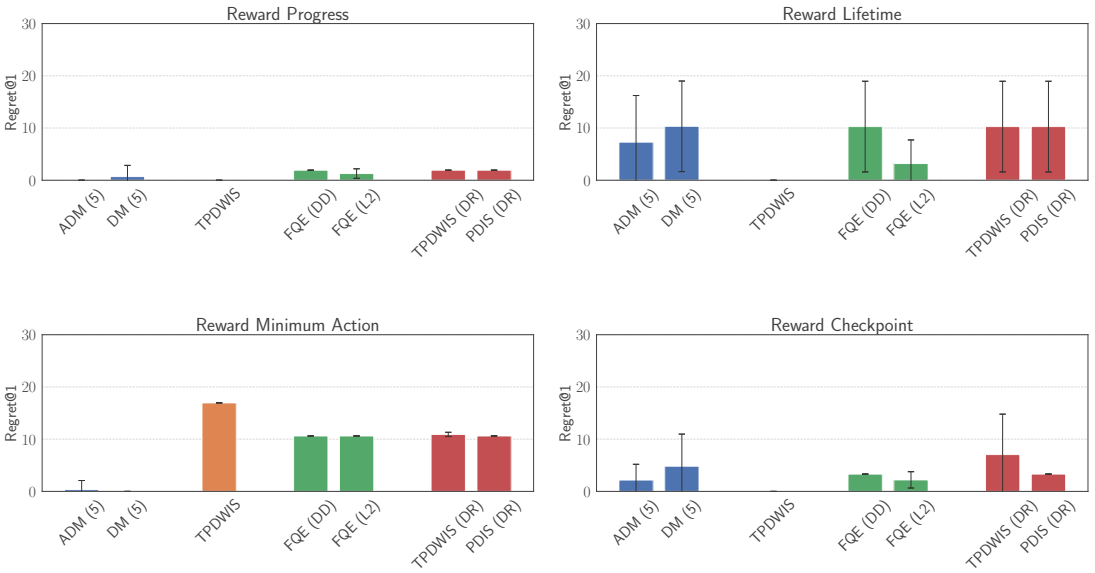


Figure 5.16: R@1 for our different OPE methods and the rewards investigated.

the sparse Checkpoint Reward presents more of a challenge for all OPE methods, as indicated by lower performance metrics. The long-horizon sparse Lifetime Reward proves to be the most difficult, with all estimators yielding relatively low SRC. Additionally, while all other estimators perform well on the Minimum Action Reward, our IS method struggles with this metric. Again, we hypothesise that the reason might be the evaluation procedure we had to adopt for IS.

5.4 Which OPE method should be use?

We will give a short summary of OPE selection criteria concerning the following four metrics:

- Performance
- Ease of Use
- Required Compute
- Interpretability

At the end of this section, we will summarize our recommendations for selecting an OPE method.

Performance: Generally, ADM (5) appears to perform best across most of our rewards and metrics. This is closely followed by the FQE and DR methods. However, employing DR in our setting results in very similar outcomes to plain FQE, with significant performance increases only for the sparse Checkpoint Reward and slight decreases in performance in many other cases. TPDWIS performs competitively across most metrics and rewards, except for the Minimum Action Reward, where it is significantly outperformed by all other methods. One reason for this underperformance might be the evaluation procedure we adopt for our IS estimators.

Ease of use: IS methods can be considered the easiest to use, as they do not require any engineering effort specific to the dataset. Given a dataset of transitions, we simply need trajectories with annotated rewards for each transition and the probabilities associated with the actions taken in the trajectory. In contrast, both FQE and MB methods have a much larger design space, as neural network architecture, loss function, and hyperparameters must be decided. Overall, the design space is larger and more complicated for MB methods. Methods directly taken from the literature yield worse results in our benchmark than FQE, and only specific engineering efforts enable MB to become competitive and outperform FQE.

Required compute: IS methods require very little compute, as we only need to calculate the IS-ratio for each evaluation agent. This is followed by MB methods, where we perform one training run with our behavior dataset and then can perform rollouts and reward estimation with one trained model for all our evaluation agents. FQE requires the most compute by a significant margin, as we need a separate training run for each evaluation agent.

Interpretability: It can be argued that MB methods are the most interpretable, eventhough they employ a black-box neural network, as their trajectory rollouts can be visualized, as is commonly done in simulation. In IS, we can identify trajectories that are similar to our target agent, enabling a certain degree of interpretability as well. FQE suffers from the worst interpretability, as it provides a singular performance estimate for one starting point without any insights into how this estimate was derived.

To sum up, for the F1TENTH environment, MB methods require the highest amount of engineering effort. Still, they typically result in the best-performing techniques, with high interpretability and low compute requirements. FQE methods require less engineering effort and yield well-performing methods but suffer from low interpretability and high compute requirements. IS methods require virtually no engineering effort, result in generally good-performing estimators, offer some interpretability, and have very low compute requirements. DR methods offer little advantage over plain FQE in our environment but come with all the associated drawbacks.

CHAPTER 6

Conclusion

In the future, OPE methods are envisioned to reduce the required resources to develop autonomous systems and increase the safety of deployed systems. By applying OPE to a real-world robotics scenario, this thesis bridges a crucial gap between previous investigations of OPE in simulation environments and practical real-world robotics applications, bringing this vision closer to reality. Specifically, utilizing the F1TENTH racing environment, this thesis introduces the first real-world robotics OPE benchmark. This novel benchmark is not only a significant step toward validating OPE methods in real-world settings but also provides a framework for evaluating the effectiveness of novel OPE methods in general.

The specific contributions made through this thesis include the following:

1. **F1TENTH Offline RL Dataset:** A novel real-world dataset with 55 unique autonomous racing policies, exhibiting a wide variety of behaviors, is introduced.
2. **Benchmark and Adaption of OPE Methods:** We provide a comprehensive benchmark of over 20 OPE methods, adapting these methods to our novel dataset as necessary. By utilizing and modifying techniques from existing literature, we demonstrate how high-performing MB methods can be designed. Specifically, the combination of AM with DM results in a significant improvement in performance.
3. **Novel Estimator:** We introduce the Termination-aware Per-Decision-Weighted Importance Sampling (TPDWIS) estimator, the only IS estimator competitive with MB and FQE methods, and formally prove its consistency. The good performance of TPDWIS is due to the explicit handling of early termination trajectories.
4. **Analysis of OPE Performance:** We show that different rewards affect OPE performance and identify the best-performing methods across multiple metrics. Furthermore, we provide guidelines for selecting OPE methods depending on

the use case, specifically for the F1TENTH environment, with expected (partial) transferability to other environments.

Despite significant contributions, some limitations remain: First, in relation to our investigated setting, we only study a single racing track and one mobile robot, limiting the transferability of findings to other robots or configurations. Since our research is focused on a single environment, the examination of the novel TPDWIS estimator is limited, restricting our ability to make broader comparisons with previous approaches that take early terminations into account. Furthermore, we assumes relatively dense support in the training set, which may not be representative of all potential use cases. Secondly, we do not investigate the impact of our evaluation procedure on the performance of IS, a potential reason for IS generally being outperformed by other estimators. Last, a low number of seeds used in evaluating FQE methods may affect the robustness of the findings in relation to this method.

We identify the following interesting future research directions:

1. **Real-World Benchmarking Framework:** We envision a plug-and-play framework for investigating and benchmarking the performance of novel OPE methods across multiple real-world environments. Such a benchmarking framework, in the contrast to the current one, should include the following: First, as previous studies have shown that the specific environment has a significant impact on the performance of specific OPE methods, additional datasets and environments should be provided in the framework. One such environment could be the existing real-world Trifinger dataset [WWG+21]. Second, the framework should include well tuned MB and FQE-based methods for all environments. Third, the user should be provided with control over the sparsity of the off-policy data presented to the OPE algorithm. Finally, the framework should be easily extensible to new methods.
2. **Cross-Robot Transferability:** To generalize findings across different robots of the same kind, the framework should include datasets and ground truths from multiple units of the same robot type. This approach will help assess how OPE methods handle variations within the same category of robots, such as manufacturing inconsistencies, differing sensor noise, and wear. Hopefully, the availability of such datasets will enable OPE methods to give performance estimates not just for specific robots, but generalize across robots of the same kind. This focus on cross-robot transferability will enhance the practical applicability of OPE techniques, ensuring they remain effective even when utilized in real-world scenarios and datasets of the particular robot are not available.

In conclusion, this study demonstrates how OPE can be applied to real-world robotics platforms, potentially enabling safer and less expensive deployments. We not only introduce the first real-world robotics benchmark for OPE but also presents the novel and performant TPDWIS estimator. Our findings will inform the future development and deployment of safer robotics algorithms in real-world applications.

List of Figures

List of Tables

List of Algorithms

Acronyms

ADM Autoregressive Delta Model. 40, 41, 57, 63, 74, 75, 77

AM Autoregressive Model. 17, 40, 41, 57, 63, 74, 79

DM Delta Model. 39–41, 63, 79

DM (LL) Delta Model (Log-Likelihood). 57, 74

DM (MSE) Delta Model (Mean-Squared-Error). 57, 63

DR Doubly Robust Estimator. 16, 17, 35, 52, 69–72, 74, 75, 77, 83

F1TENTH F1TENTH. xi, xiii, xv, 3, 19–21, 23, 25, 27, 32, 33, 35, 45, 55, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 77, 79–81, 85

FQE Fitted-Q-Evaluation. 9, 11, 15–17, 27, 35, 42–44, 52, 63–65, 70–72, 74, 75, 77, 79, 80, 83

FQE-DD Fitted-Q-Evaluation with discrete distribution. 17, 43, 52, 63, 65, 69, 70, 83

FQE-L2 Fitted-Q-Evaluation with L2 loss. 17, 42, 43, 63, 65, 75, 83

FTG Follow-the-Gap. 26–29, 32, 37, 45, 59, 60, 62, 82, 85

IS Importance Sampling. xi, xiii, xv, 9, 11–17, 27, 35, 44–52, 65–72, 74–77, 79–83

LL Log-Likelihood. 35, 38–41, 63

MAE Mean Absolute Error. 57, 59, 63, 66, 67, 69–71, 74, 75, 82, 83

MB Model-based. 9, 11, 15, 17, 35, 37, 39, 42, 44, 57–60, 74, 75, 77, 79, 80, 82, 85

MDP Markov Decision Process. 5–7, 11, 13, 14, 19, 21, 23, 24, 45, 81

MSE Mean-Squared-Error. 35, 37–39, 41, 63

NM (LL) Naive Model (Log-Likelihood). 57

NM (MSE) Naive Model (Mean-Squared-Error). 57, 63

OPE Off-Policy Evaluation. xi, xiii, xv, 2, 3, 5, 7–11, 13, 16, 17, 19, 22, 24–26, 31, 35, 38–40, 52, 55–59, 74–77, 79–83

PDIS Per-Decision Importance Sampling. 14, 15, 17, 45, 47, 52, 65, 67, 74

PDWIS Per-Decision-Weighted Importance Sampling. 17, 44, 47–50, 65, 69, 70, 74

PHWIS Per-Horizon Weighted Importance Sampling. 44, 46–48, 50, 65, 67, 68

PP Pure Pursuit. 26, 29, 30, 32, 45, 59, 62, 82, 85

R@1 Regret@1. 56–58, 63, 67, 69, 72, 74–76, 82, 83

RL Reinforcement Learning. xi, xiii, 1, 2, 11, 17, 19, 26, 40, 42, 43, 79

SRC Spearman Rank Correlation. 56–58, 63–67, 69, 70, 74–76, 82, 83

TPDWIS Termination-aware Per-Decision-Weighted Importance Sampling. xi, xiii, 3, 35, 44, 45, 47, 49–52, 66, 69, 70, 74, 75, 77, 79, 80

WIS Weighted Importance Sampling. 15, 17, 45–47, 51, 52, 65, 67, 70, 74

Bibliography

[ACML18] Kavosh Asadi, Evan Cater, Dipendra Misra, and Michael L. Littman. Towards a simple approach to multi-step model-based reinforcement learning. *arXiv preprint arXiv:1811.00128*, 2018.

[AN04] Pieter Abbeel and Andrew Ng. Learning first-order markov models for control. *Advances in Neural Information Processing Systems*, 17, 2004.

[BB20] Varundev Suresh Babu and Madhur Behl. f1tenth. dev-an open-source ros based f1/10 autonomous racing simulator. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1614–1620. IEEE, 2020.

[BBB+22] Axel Brunnbauer, Luigi Berducci, Andreas Brandstátter, Mathias Lechner, Ramin Hasani, Daniela Rus, and Radu Grosu. Latent imagination facilitates zero-shot transfer in autonomous racing. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7513–7520. IEEE, 2022.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[BDM17] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[BMHB+18] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, TB Dhruva, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *International Conference on Learning Representations*, 2018.

[BZL+22] Johannes Betz, Hongrui Zheng, Alexander Liniger, Ugo Rosolia, Phillip Karle, Madhur Behl, Venkat Krovi, and Rahul Mangharam. Autonomous vehicles on the edge: A survey on autonomous vehicle racing. *IEEE Open Journal of Intelligent Transportation Systems*, 3:458–488, 2022.

[CCML18] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in Neural Information Processing Systems*, 31, 2018.

[CLW+07] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. Usarsim: a robot simulator for research and education. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405. IEEE, 2007.

[DTB18] Shayan Doroudi, Philip S. Thomas, and Emma Brunskill. Importance sampling for fair policy selection. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5239–5243. International Joint Conferences on Artificial Intelligence Organization, 2018.

[EEJ21a] Benjamin Evans, Herman A Engelbrecht, and Hendrik W Jordaan. Learning the subsystem of local planning for autonomous racing. In *2021 20th International Conference on Advanced Robotics (ICAR)*, pages 601–606. IEEE, 2021.

[EEJ21b] Benjamin Evans, Herman A Engelbrecht, and Hendrik W Jordaan. Reward signal design for autonomous racing. In *2021 20th International Conference on Advanced Robotics (ICAR)*, pages 455–460. IEEE, 2021.

[F1T23] F1TENTH Foundation. Build your own f1tenth car. `https://f1tenth.org/build.html`, 2023. Accessed: 2024-03-18.

[Far23] Farama Foundation. Highwayenv: An environment for autonomous driving and tactical decision-making tasks. `https://github.com/Farama-Foundation/HighwayEnv/blob/master/scripts/parking_model_based.ipynb`, 2023. Accessed: 2024-07-16.

[FKN+20] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[FNN+21] Justin Fu, Mohammad Norouzi, Ofir Nachum, George Tucker, Ziyu Wang, Alexander Novikov, Mengjiao Yang, Michael R Zhang, Yutian Chen, Aviral Kumar, et al. Benchmarks for deep off-policy evaluation. *arXiv preprint arXiv:2103.16596*, 2021.

[GB24] Radu Grosu and Andreas Brandstätter. Autonomous racing cars, 2024. Course Number 191.119, TU Wien.

[GBD92] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[GWN+20] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Thomas Paine, Sergio Gómez, Konrad Zolna, Rishabh Agarwal, Josh S Merel, Daniel J Mankowitz, Cosmin Paduraru, et al. Rl unplugged: A suite of benchmarks for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:7248–7259, 2020.

[Hok15] Hokuyo Automatic Co., Ltd. *UST-10LX Scanning Laser Rangefinder Specifications*, 2015.

[JL16] Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 652–661. PMLR, 2016.

[JWH+13] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. *An introduction to statistical learning*, volume 112. Springer, 2013.

[KN20] Ilya Kostrikov and Ofir Nachum. Statistical bootstrapping for uncertainty estimation in off-policy evaluation. *arXiv preprint arXiv:2007.13609*, 2020.

[LKTF20] 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.

[LVY19] Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR, 2019.

[MBJ17] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Learning multimodal transition dynamics for model-based reinforcement learning. *arXiv preprint arXiv:1705.00470*, 2017.

[MBP+23] 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.

[MJ21] Steve Macenski and Ivona Jambrecic. Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6(61):2783, 2021.

[Ott19] N. Otterness. The "disparity extender" algorithm, and f1/tenth. `https://www.nathanotterness.com/2019/04/the-disparity-extender-algorithm-and.html`, 2019. Accessed: 2024-03-27.

[OZKM20] Matthew O'Kelly, Hongrui Zheng, Dhruv Karthik, and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of Machine Learning Research*, 123, 2020.

[PPM+20] Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

[PSS00] Doina Precup, Richard S Sutton, and Satinder P Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 759–766, 2000.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[QZG+22] Rong-Jun Qin, Xingyuan Zhang, Songyi Gao, Xiong-Hui Chen, Zewen Li, Weinan Zhang, and Yang Yu. Neorl: A near real-world benchmark for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 35:24753–24765, 12 2022.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[SFMP21] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access*, 9:153171–153187, 2021.

[SG12] Volkan Sezer and Metin Gokasan. A novel obstacle avoidance algorithm:"follow the gap method". *Robotics and Autonomous Systems*, 60(9):1123–1134, 2012.

[SMGDV21] Simon P Shen, Yecheng Ma, Omer Gottesman, and Finale Doshi-Velez. State relevance for off-policy evaluation. In *International Conference on Machine Learning*, pages 9537–9546. PMLR, 2021.

[Spa19] SparkFun Electronics. 9dof razor imu m0 firmware and documentation. `https://github.com/sparkfun/9DOF_Razor_IMU`, 2019. Accessed: 2024-02-15.

[Sta12] Problem with unbiased but not consistent estimator. `https://math.stackexchange.com/questions/119461/problem-with-unbiased-but-not-consistent-estimator`, 2012. Accessed: 2024-03-31.

[TB16] Philip S. Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. *CoRR*, abs/1604.00923, 2016.

[Tho15] Philip Thomas. *Safe Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 2015.

[TUM23] TUMFTM. Global race trajectory optimization. https://github.com/TUMFTM/global_racetrajectory_optimization, 2023. Accessed: 2024-07-16.

[US21] Masatoshi Uehara and Wen Sun. Pessimistic model-based offline reinforcement learning under partial coverage. In *International Conference on Learning Representations*, 2021.

[VLA+21] Cameron Voloshin, Hoang M Le, AI Argo, Nan Jiang, and Yisong Yue. Empirical study of off-policy policy evaluation for reinforcement learning. 2021.

[WK18] Corey H Walsh and Sertac Karaman. Cddt: Fast approximate 2d ray casting for accelerated localization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3677–3684. IEEE, 2018.

[WST+85] Richard S Wallace, Anthony Stentz, Charles E Thorpe, Hans P Moravec, William Whittaker, and Takeo Kanade. First results in robot road-following. In *IJCAI*, volume 2, pages 1089–1095, 1985.

[WWG+21] Manuel Wuthrich, Felix Widmaier, Felix Grimminger, Shruti Joshi, Vaibhav Agrawal, Bilal Hammoud, Majid Khadiv, Miroslav Bogdanovic, Vincent Berenz, Julian Viereck, et al. Trifinger: An open-source robot for learning dexterity. In *Conference on Robot Learning*, pages 1871–1882. PMLR, 2021.

[ZPN+21] Michael R Zhang, Thomas Paine, Ofir Nachum, Cosmin Paduraru, George Tucker, Mohammad Norouzi, et al. Autoregressive dynamics models for offline policy evaluation and optimization. In *International Conference on Learning Representations*, 2021.

[ZQW20] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020.