ABSTRACT

Offline reinforcement learning (RL) has gained significant attention in recent years as a promising approach for training intelligent agents without the need for real-time interaction with an environment.

This capability addresses a significant limitation in traditional reinforcement learning, where direct interaction with environments is required, limiting its widespread adoption, especially in domains where such interactions are dangerous or impractical. With offline reinforcement learning, the potential for creating autonomous agents extends to various fields where direct environment interaction is challenging, enabling safer and more efficient deployment in areas such as healthcare, finance, and hazardous environments.

Successful implementations of offline reinforcement learning could derive optimal policies from the given data, facilitating the automation of diverse decision-making realms, including but not limited to healthcare, education, and robotics.

In this thesis, we explore the application of offline RL techniques in the context of World of Tanks, a popular online multiplayer tank combat game.

This study evaluated several offline RL algorithms, including Conservative Q-Learning (CQL), Implicit Q-Learning (IQL), Decision Transformer (DT), and Deep Deterministic Policy Gradient (DDPG). The results showed that CQL and IQL achieved significant returns under various discount factors, demonstrating robustness and adaptability in offline settings. The choice of discount factor (γ) had a notable impact on the algorithms' performance, with higher discount factors leading to better cumulative returns, particularly for CQL and IQL. Handling data distribution shifts is crucial for the robustness of offline RL policies; techniques such as regularization

in CQL and modified architectures in IQL were effective in mitigating the effects of distribution shifts. The volume of training data significantly influenced the performance of offline RL algorithms, with larger datasets improving learning effectiveness. The study also highlighted the inherent challenges in evaluating offline RL policies due to the absence of real-time interaction with the environment. Methods such as model-based dynamics and policy value estimation were employed, though they have limitations in accurately predicting real-world performance.

Ioannis Pastellas – University of Cyprus, 2024

The abstract is **not** paginated.

OFFLINE REINFORCEMENT LEARNING IN WORLD OF TANKS

Ioannis Pastellas

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

June, 2024

APPROVAL PAGE

Master of Science Thesis

OFFLINE REINFORCEMENT LEARNING IN WORLD OF TANKS

Presented by

Ioannis Pastellas

Research Supervisor _____
Chris Christodoulou

Committee Member _____
Vassilis Vassiliadeis

Committee Member _____
Andreas Aristidou

University of Cyprus

June, 2024

ACKNOWLEDGEMENTS

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Dr. Chris Christodoulou, and Dr. Vassilis Vassiliadeis for his continuous support, guidance, and invaluable insights throughout the course of this research. Their expertise and encouragement have been instrumental in the completion of this thesis.

I would also like to extend my appreciation to my committee members, and Dr. Andreas Aristidou, for their valuable feedback and suggestions which significantly improved the quality of this work.

A special thanks to Wargaming Group LTD for providing the necessary data and support, which was crucial for the development and in this thesis. Their insights and continuous support, was as valuable as my supervisors, being all together in this work.

I am deeply grateful to my family and friends for their unwavering support and understanding during the challenging times of this research journey. Their patience and encouragement have been a constant source of motivation.

Lastly, I would like to thank my colleagues and the faculty members at the University of Cyprus for creating an intellectually stimulating environment that fostered my knowledge growth.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

World of Tanks is a popular online multiplayer game where players command historically accurate tanks in team-based battles. The game's strategic and tactical aspects make it an ideal environment for exploring reinforcement learning techniques. However, training RL agents in such an online setting can be challenging due to the need for real-time interaction, which often requires substantial computational resources and can disrupt the gaming experience for human players, or it is not feasible at all due to other reasons.

Offline reinforcement learning, also known as batch reinforcement learning, offers an alternative approach by learning from a static dataset of previously collected experiences. This thesis aims to investigate the potential of offline RL in training intelligent tank AI agents for tactical decision-making in World of Tanks.

1.2 World of Tanks Game

”World of Tanks” [1], developed by Wargaming Group Limited, is set in the historical context of World War II and the Cold War era, with a primary focus on armored warfare. The game features an extensive collection of accurately modeled tanks and armored vehicles from various nations involved in these conflicts. This historical backdrop serves as the foundation for the game’s immersive and authentic tank battles.

”World of Tanks” offers players the opportunity to command a wide array of tanks, each with distinct characteristics, strengths, and weaknesses. Players can choose from various tech trees, each representing a different nation’s tanks. Tanks can be further customized with upgrades and equipment, allowing players to fine-tune their vehicles to suit their preferred playstyle.

The core gameplay of ”World of Tanks” revolves around team-based battles where two teams of players, typically consisting of 15 members each, engage in strategic tank warfare. Players must work together to achieve objectives, depending on the gamemode, such as capturing flags or eliminating enemy tanks, while also considering terrain, cover, and their tank’s capabilities.

The gameplay in ”World of Tanks” is characterized by its tactical complexity. Players must employ a combination of strategic planning, map awareness, and tank-specific knowledge to succeed. The vast selection of tanks, each with unique attributes, such as health of tank etc, which adds layers of complexity to decision-making. This complexity provides a challenging environment for reinforcement learning agents to navigate and learn from.

1.3 Objectives

1. **Assessing Offline RL Algorithms**: The primary objective was to evaluate the performance of different offline RL algorithms within the context of the World of Tanks game. This involved assessing how well these algorithms could learn from a fixed dataset without additional online interactions.

2. **Impact of Key Parameters**: Another objective was to investigate the impact of key parameters, such as discount factors and dataset size, on the performance of offline RL algorithms.

3. **Algorithm Comparison**: The study aimed to compare the performance of standard RL algorithms like DDPG with specialized offline RL algorithms such as CQL and IQL to identify which approaches are more effective in offline settings.

1.4 Contributions

1. **Demonstration of Offline RL Feasibility**: The study demonstrated that offline RL algorithms could effectively learn and perform well in complex game environments like World of Tanks, which traditionally require real-time interactions for training.

2. **Algorithm Insights**: Provided detailed insights into the strengths and weaknesses of various offline RL algorithms, particularly highlighting the robustness of CQL and IQL in handling data distribution shifts and achieving high returns.

3. **Evaluation Framework**: Developed a comprehensive framework for evaluating offline RL policies using policy value estimation and especially model-based dynamics , contributing to the methodology of offline RL research.

4. **Recommendations for Future Research**: Suggested directions for future research, including enhancing exploration techniques, improving dataset quality, adapting to concept drift, and developing multi-agent learning approaches for better team coordination in games.

Chapter 2

Background

2.1 Related Work

2.1.1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning concerned with how agents ought to take actions in an environment to maximize cumulative reward. This learning paradigm is inspired by behavioral psychology, where an agent learns to achieve goals through trial and error, receiving feedback from its own actions and experiences.

2.1.1.1 Core Concepts of Reinforcement Learning

At the heart of RL are several key concepts:

- **Agent**: The learner or decision maker that interacts with the environment.
- **Environment**: Everything the agent interacts with and operates within.
- **State** (s): A representation of the current situation or configuration of the environment.

- **Action** (a): A decision or move made by the agent.
- **Reward** (r): Feedback from the environment in response to an action taken by the agent.
- **Policy** (π): A strategy or mapping from states to actions, defining the agent's behavior.
- **Value Function**: A prediction of future rewards used to evaluate the goodness of states or actions.

2.1.1.2 Key Algorithms in Reinforcement Learning

Several algorithms have become fundamental to RL, each contributing uniquely to the field:

- **Q-Learning**: Proposed by Watkins in 1989 [38], Q-Learning is a model-free algorithm that seeks to learn the value of an action in a particular state. It uses the Bellman equation to update Q-values iteratively, ensuring convergence to the optimal policy. A lot of Offline RL algorithms are based on Q-Learning.
- **SARSA (State-Action-Reward-State-Action)**: Unlike Q-Learning, SARSA updates its Q-values based on the action actually taken by the policy, making it an on-policy method. This allows SARSA to learn more conservative policies suitable for specific applications where safety is a concern.
- **Deep Q-Networks (DQN)**: Introduced by Mnih et al. in 2015 [22], DQN combined Q-Learning with deep neural networks, enabling RL to scale to high-dimensional state spaces like those encountered in Atari game environments. This marked a significant milestone, demonstrating the capability of RL to handle complex tasks.

- **Policy Gradient Methods**: These methods, including REINFORCE and Actor-Critic algorithms, directly parameterize the policy and optimize it using gradient ascent. This approach is particularly useful for problems with continuous action spaces or where value-based methods struggle.

2.1.1.3 Applications of Reinforcement Learning

RL has been successfully applied across various domains:

- **Game Playing**: RL algorithms have achieved superhuman performance in games [30] like chess, Go, and various Atari games. AlphaGo, developed by DeepMind, is a prominent example, defeating the world champion in Go using a combination of RL and Monte Carlo Tree Search (MCTS). More over, it is succesfully used in worldwide video games like Minecraft

- **Robotics**: In robotics, RL is used for training autonomous agents to perform tasks such as navigation, manipulation, and locomotion. Robots learn to adapt to dynamic environments and execute complex sequences of actions [32].

- **Healthcare**: Personalized treatment plans and drug discovery are areas where RL shows promise. By modeling patient responses and optimizing treatment sequences, RL can potentially improve outcomes and efficiency in healthcare delivery [19].

2.1.1.4 Markov Decision Process (MDP)

A Markov Decision Process (MDP) is a mathematical framework used in reinforcement learning to model decision-making problems. It consists of the following components:

- **States ($\mathcal{S}$)**: A finite set of possible situations or configurations that the agent can be in.
- **Actions ($\mathcal{A}$)**: A finite set of possible actions that the agent can take.
- **Transition Probability Function (P)**: $P(s_{t+1}|s_t, a_t)$ is the probability that taking action a_t in state s_t will lead to state s_{t+1}.
- **Reward Function (R)**: $R(s_t, a_t)$ is the immediate reward received by the agent for taking action a_t in state s_t.
- **Discount Factor (γ)**: A value between 0 and 1 representing the preference for current rewards over future rewards.

The goal in MDPs is to find a policy π, which is a mapping from states to actions, that maximizes the expected cumulative reward over time.

2.1.2 Offline Reinforcement Learning

With the widespread adoption of deep learning, reinforcement learning (RL) has gained significant popularity, tackling previously insurmountable challenges such as playing complex games from pixel observations, engaging in sustained conversations with humans, and controlling robotic agents, as mentioned before. Despite these advancements, many domains remain inaccessible to RL due to the high costs and dangers associated with interacting with the environment

Offline RL, also known as batch RL, is a data-driven RL paradigm that is used for tackling the above issue. Unlike the classical RL, the learning process is based exclusively on static datasets comprised of previously collected experiences. In this model, a behavior policy interacts

with the environment to gather experiences (or experiences are already collected), which are subsequently used to learn a new policy without any further interaction.

This approach is particularly valuable in contexts where online interaction is impractical—such as in robotics, education, healthcare, and autonomous driving—due to the high costs or dangers associated with real-time data collection. Additionally, even in scenarios where online interaction is feasible, utilizing previously collected data can offer enhanced generalization across complex domains. In addition to that, Offline RL can utilize the huge dataset already available or already pre-trained Models. Offline RL distinguishes itself by learning from a static dataset without ongoing updates from environment interaction, unlike online and off-policy RL which continually adapt their policies through direct engagement. After an offline policy is developed, there remains the option to fine-tune this policy through online methods, providing a safer and potentially less costly initial approach compared to starting with a random policy.

Another common term is ”batch reinforcement learning” [28]. For clarity purposes, this work uses the term ”offline reinforcement learning.”

For more formal and more thorough, survey, on definition, problems, algorithms and applications, [27] and [18] are a great start.

2.1.3 Offline Reinforcement Learning Process

In offline reinforcement learning (RL), we work with a given static dataset to train a policy, rather than interacting with the environment in real-time.

Formal Definition

More formally, the offline dataset $\mathcal{D}$ consists of tuples (s, a, s', r) sampled from some unknown policy π_β, often referred to as the behavior policy. It is important to note that π_β may represent a mixture of different policies. For example, in the setting of World of Tanks, there is a mixture of different policies, as the data consists of episodes derived from a lot of different players.

The state s is sampled from a distribution $\mathcal{D}^{\pi_\beta}(\cdot)$, the action a is sampled according to the behavior policy $\pi_\beta(\cdot|s)$, the subsequent state s' follows the state transition probability $p(\cdot|s, a)$, and the reward r is a function of the current state and action, $r(s, a)$.

Objective

The objective in offline RL is the same as the standard RL, and is to maximize the expected sum of rewards:

$$\max_\theta \sum_t E_{s_t \sim \mathcal{D}^{\pi_\beta}(\cdot), a_t \sim \pi_\theta(\cdot|s_t)} \left[r(s_t, a_t) \right]$$

2.1.4 Offline RL Properties

Offline reinforcement learning (RL) exhibits several distinctive properties that differentiate it from traditional online RL settings:

- **Static Data Source:** The learning algorithm operates entirely on a pre-collected, static dataset. Unlike online RL, the agent cannot interact with the environment to collect new

data, which limits its ability to adapt based on new experiences. However, it is common that both offline and online interaction is used when online interaction is possible.

- **Absence of Exploration-Exploitation Trade-off:** In offline RL, the traditional exploration-exploitation dilemma, which involves deciding whether to explore new actions or exploit known rewarding ones, is not applicable. Instead, the challenge is to extrapolate or generalize from the existing data to unseen situations.

- **Data Stiching** Effective offline RL methods have the capability to ”stitch together” good behaviors from disparate parts of the dataset. Even if the overall policy that collected the data was suboptimal, these methods can extract and combine the most effective actions from different contexts within the data.

- **Derivation of Novel Actions:** A crucial property of offline RL is its goal to derive actions that differ from those recorded in the dataset. This is essential because achieving superior performance often requires the policy to deviate from the behavior previously exhibited by the data-collecting agent. Offline RL algorithms, with the help of Data Stiching. must leverage the existing data to infer potentially better actions that were not explored during data collection, essentially combining different behaviours from the dataset.

These properties underscore the unique challenges and opportunities presented by offline RL, requiring specialized algorithms and approaches to effectively learn from static datasets without direct interaction with the environment.

2.1.5 Challenges Problems in Offline Reinforcement Learning

While Reinforcement Learning (RL) represents a critical frontier in the quest for creating intelligent systems, offering the potential to empower machines with the ability to learn solely from data, however the field faces a multitude of open problems and challenges that stem from its inherent complexity and the diversity of potential applications. Key among these challenges is the distributional shift between the data available in the offline dataset and the policy being learned, which can lead to significant performance degradation if not properly addressed. Additionally, Offline RL must contend with the issues of data diversity and coverage, as the quality and comprehensiveness of the pre-collected dataset directly impact the effectiveness of the learned policy. Another critical challenge is the evaluation and validation of policies without the luxury of trial-and-error in the environment, necessitating the development of robust evaluation methodologies that can accurately predict real-world performance. Moreover, the integration of domain knowledge and constraints into the learning process remains a complex problem, essential for ensuring the safety and feasibility of deployed policies. Addressing these challenges requires innovative solutions and advancements in algorithmic techniques, making Offline RL a fertile ground for research with the potential to significantly broaden the applicability of reinforcement learning in the real world. Below there is more analytical description of open problems [18, 27, 14].

2.1.5.1 Distribution Shift

Suppose, since we are using static dataset, use the standard off-policy algorithms and objectives. Consider the

$$L(\theta) = \sum_{(s,a,s')\sim\mathcal{D}} \Bigg[(\hat{Q}_\theta(s,a) - \Big(r(s,a) + \gamma \max_{a' \in A} \hat{Q}_\theta(s',a') \Big) \Big)^2], \tag{1}$$

The problem is where a' is not Present in the D. This can cause the following

- The policy may be unreliable on out-of-distribution (OOD) actions.
- There is a tendency for the policy to seek out actions where the value function is over-optimistic.
- After values propagate, the estimated Q-values can become substantially overestimated.

Below, it is more visible the overestimation of value due to distribution shift. In the figure 1, the blue line represents the true (unknown) Q function, while the orange line shows the current estimation based on the recorded data points (green dots). The estimated Q function is accurate in areas dense with data points but tends to deviate from the true Q in regions with sparse data, leading to both positive and negative errors.

The critical point is that if new points are selected according to a policy similar to equation (1), the selection tends to focus on points where the error from the true Q has the largest positive value. This approach misses the true maximum points, but more importantly, it leads to selecting points with high positive errors.

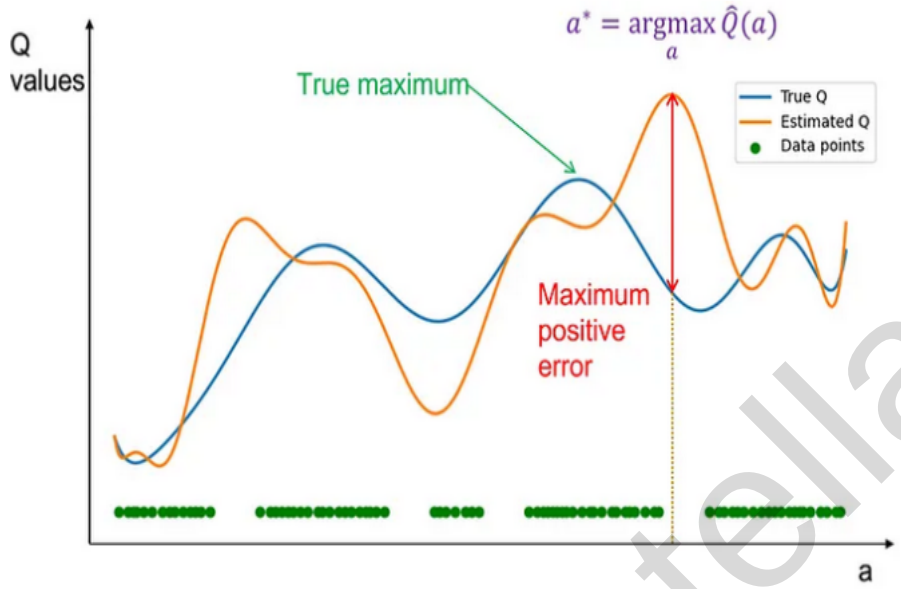


Figure 1: Illustration of Overestimation issue in Offline RL. Q-Value on actions not appearing in dataset (green dots) have higher estimated value than their true value. This results to erroneous selection of actions that are not optimal in the true Q-function context. [14]

2.1.5.2 Confounding Variables

Hidden confounding introduces a significant bias in the estimation of policies from offline data. Since the decision-making process and the outcomes are influenced by factors not accounted for in the data, traditional offline RL methods can lead to incorrect inferences about the effectiveness of certain actions. This can result in the learning of suboptimal policies that, when deployed, perform poorly or even dangerously in the real world. The issue is particularly pronounced in sequential decision-making tasks, where the impact of confounding variables can accumulate over time, leading to increasingly larger errors as the horizon extends.

For example, in autonomous driving, the decision-making process of the observational policy might be influenced by unobserved factors such as road conditions. These hidden variables affect both the actions taken by the driver and the subsequent outcomes, such as the safety and efficiency of the driving behavior. Similarly, in healthcare settings, decisions made by physicians could be influenced by unrecorded patient information, such as socio-economic status or subtle visual cues, which also affect patient outcomes. These unobserved factors create a disconnect between the recorded data and the actual decision-making process, making it challenging to accurately evaluate and improve policies based on offline data.

A great work, that takes into consideration this issue, is presented at [24].

2.1.6 Offline Reinforcement Learning Algorithms

2.1.6.1 Behavioral Cloning

Behavioral Cloning, often referred to as ”imitation learning,” attempts to replicate the behavior of an expert in a supervised learning manner. By learning from the expert's behavior, Behavioral Cloning (BC) aims to mimic their actions and decision-making processes. In the context of this thesis and applications such as World of Tanks, BC endeavors to learn player policies and behavior using a collected dataset. Since BC focuses on imitating action distributions, its performance will be close to the mean behavior observed in the dataset.

In a continuous action scenario, BC must model a probability distribution over actions conditioned on the state. To achieve this, a softmax activation function is employed in the output

layer of the Behavioral Cloning network. This allows the network to generate a probability distribution over possible actions for a given state, aligning with common practices in policy-based reinforcement learning methods.

Let $\pi_\theta(s_t)$ represent the policy network's prediction of action given the state s_t. The mean squared error (MSE) loss is used to measure the dissimilarity between the predicted continuous action $\pi_\theta(s_t)$ and the expert actions. The equation for the MSE loss is as follows:

$$L(\theta) = E_{a_t, s_t \sim D} \left[(a_t - \pi_\theta(s_t))^2 \right]$$

Where:

- a_t represents the expert action at time t.
- $\pi_\theta(s_t)$ is the policy network's prediction of the action given the state s_t.
- D is the dataset of state-action pairs.

The goal is to minimize this MSE loss, encouraging the BC network to produce action predictions that closely match the expert actions for each state.

2.1.6.2 Conservative Q-Learning

CQL stands as a data-driven deep reinforcement learning algorithm rooted in the Soft Actor-Critic (SAC) framework [16]. It excels in addressing offline RL challenges, consistently delivering top-tier performance.

CQL addresses overestimation by explicitly penalizing the Q-value estimates for actions not present in the dataset. This penalty discourages the value function from assigning high values

to unseen actions, thus reducing overestimation bias. The CQL objective can be formulated as follows:

$$\hat{Q}^{\pi}_{\text{CQL}} \leftarrow \arg\min_{Q} \max_{\mu(a|s)} \Big(\alpha E_{s\sim\text{data},a\sim\mu(a|s)}[Q(s,a)] - \alpha E_{s,a\sim\text{data}}[Q(s,a)]\Big) + E_{s,a,s'\sim\text{data}}\left[\Big(r(s,a) + \gamma E_{\pi}[\hat{Q}(s',a')] - Q(s,a)\Big)^2\right] \tag{2}$$

The main idea behind CQL is to learn Q-functions (value functions) that ensure the expected value of a policy is not overestimated, and thus it should at least be as good as the true value of the policy.

Breaking down the equation, we have:

- **Outer Objective**: $\hat{Q}^{\pi}_{\text{CQL}} \leftarrow \arg\min_{Q} \max_{\mu(a|s)}$

  This indicates that CQL is trying to solve for the Q-function that minimizes the maximum expected value under some policy μ. This minimax formulation helps to find a conservative estimate of the Q-function.

- **First Term**: $\alpha E_{s\sim\text{data},a\sim\mu(a|s)}[Q(s,a)] - \alpha E_{s,a\sim\text{data}}[Q(s,a)]$

  This difference represents the CQL regularizer. The algorithm minimizes the Q-values under a chosen policy distribution μ while maximizing them under the data distribution. This term acts as a regularizer that prevents overestimation of Q-values of unseen actions. alpha , is the requlrizer variable that controls this term.

- **Second Term**: $E_{s,a,s'\sim\text{data}}\left[(r(s,a) + \gamma E_{\pi}[\hat{Q}(s',a')] - Q(s,a))^2\right]$

  This is the standard Temporal Difference (TD) error term used in Q-learning algorithms. The TD error measures the difference between the current estimate of the Q-function and

the target Q-function, which includes the reward plus the discounted value of the next state-action pair.

CQL can be to conservative at some times and in more comple real scenarios.

Despite that, CQL has demonstrated effectiveness in various offline RL tasks, significantly reducing the overestimation bias and improving the robustness and performance of the learned policies.

2.1.7 Implicit Q-Learning

Implicit Q-Learning (IQL) is an Offline Reinforcement Learning (RL) algorithm designed to estimate advantages using Temporal Difference (TD) updates without querying the Q-values on out-of-distribution (OOD) actions. This approach aims to enhance policy learning efficiency by leveraging in-distribution actions and values.

The core idea behind IQL is to estimate the value function $V(s)$ using an expectile regression loss, which allows for the adjustment of the learning process to focus on either higher or lower expectiles. The expectile regression loss is defined as follows:

$$\ell_2^\tau(x) = \begin{cases} (1-\tau)x^2 & \text{if } x < 0 \\ \tau x^2 & \text{otherwise} \end{cases}$$

where τ is a parameter controlling the expectile level. This loss function is asymmetric, providing flexibility in emphasizing different parts of the value distribution.

The IQL algorithm proceeds with the following steps:

1. **Fit $V(s)$ with expectile loss:** The value function $\hat{V}(s)$ is updated by minimizing the expectile loss, ensuring that the estimated values appropriately reflect the desired expectile level.

2. **Update $Q(s,a)$ with typical MSE loss:** The Q-values $\hat{Q}(s,a)$ are updated using the standard Mean Squared Error (MSE) loss, incorporating the rewards and the discounted estimated values of the next state.

3. **Extract policy with Advantage Weighted Regression (AWR):** The policy $\hat{\pi}(a|s)$ is derived by maximizing the expected log probability of actions weighted by the exponentiated advantages, ensuring that the policy improvement is implicit and computationally efficient.

$$\hat{V}(s) \leftarrow \arg\min_{V} E_{(s,a)\sim D}\left[\ell_2^\tau(V(s) - \hat{Q}(s,a))\right]$$

$$\hat{Q}(s,a) \leftarrow \arg\min_{Q} E_{(s,a,s')\sim D}\left[\left(Q(s,a) - \left(r + \gamma \hat{V}(s')\right)\right)^2\right]$$

$$\hat{\pi} \leftarrow \arg\max_{\pi} E_{s,a\sim D}\left[\log \pi(a|s) \exp\left(\frac{1}{\alpha}(\hat{Q}(s,a) - \hat{V}(s))\right)\right]$$

The benefits of IQL include:

- **No need to query OOD actions:** The algorithm avoids evaluating Q-values for actions outside the dataset distribution.

- **Policy training on in-distribution actions:** The policy remains robust by training solely on actions within the dataset.

- **Decoupling actor and critic training:** This separation enhances computational efficiency, making the training process faster.

This innovative approach allows IQL to effectively utilize the available data, resulting in a more efficient and robust policy learning process in offline settings.

2.1.7.1 Decision Transformer (DT)

Decision Transformer (DT) [6] is a cutting-edge approach in the field of Reinforcement Learning (RL) that has shown significant promise in addressing the challenges of offline RL. In the context of our study on Offline Reinforcement Learning DT offers a novel perspective on how agents can learn from historical data to make decisions.

In offline RL, the agent's objective is to learn a policy from a fixed dataset of past experiences without interacting with the environment. Decision Transformer, however, offers a different paradigm on addressing Offline Reinforcement and Reinforcement Learning in general. Instead of taking taking account the Markov Decision Process, where the action taken in the current state is inferred by using a value indicating the quality of the action , it views the Reinforcement Learning problem as a Sequence modelling Problem .

Thus, the core idea of DT is to model a Reinforcement Learning problem as a sequence modeling problem that progressively predict future actions, conditioned on previous states. actions, rewards and desired reward. So, the end goal of the transformer-based architecture is generate future actions that can achieve this high reward, as soon as possible.

Since the transformer should predict actions that lead to the states with the most possible future rewards instead of using past rewards the authors use rewards-to-go (the sum of the rewards for all future states starting with the current step). At test time a starting state and a target reward are provided as conditioning information. The target reward is decremented after each taken action by the achieved reward.

Mathematically, the problem is formulated as follows:

Let $\mathcal{D} = \{(\mathbf{o}_t, \mathbf{a}_t, r_t, \mathbf{o}_{t+1})\}$ be the dataset of tuples representing states, actions, rewards, and next states, respectively.

Decision Transformer (DT) leverages the power of self-attention mechanisms[36] to model sequential data efficiently.

The self-attention mechanism is a critical component in the architecture of transformer models, enabling them to weigh the importance of different tokens in a sequence when producing an output. The self-attention mechanism operates on three primary vectors: Query (Q), Key (K), and Value (V).

- **Query (Q)**: Represents the current token for which we are calculating the attention.

- **Key (K)**: Represents the tokens against which the Query is compared.

- **Value (V)**: Represents the actual values that are combined or weighted to produce the final output.

Self-Attention is given by

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3}$$

Where:

- Q is the Query matrix.

- K is the Key matrix.

- V is the Value matrix.

- d_k is the dimension of the keys (which is also typically the dimension of the queries and values).

- QK^T represents the dot product of the Query matrix with the transpose of the Key matrix.

- The result of QK^T is scaled by $\frac{1}{\sqrt{d_k}}$ to mitigate the impact of large dot product values, which can destabilize the gradients during training.

- The softmax function is applied to the scaled dot product to obtain the weights, which are then used to sum the Value matrix V.

The self-attention mechanism allows each token to focus on different parts of the input sequence dynamically, which is essential for capturing dependencies that are not necessarily local.

The difference with the basic Transformer model , is that in the case of Reinforcement Learning scenario, the input to transformer is not a sequence of words but instead a sequence of the 3-modal tuple (return-to-go, state, action). A more illustrative explanation can be seen below. In the figure, the sequence of tuples (RtG, s, a) (that serve as tokens) go through the Transformer architecture, which then outputs the next actions. It is important to note. that DT does not have the concept of Value function, so value-based evaluation cannot be used.

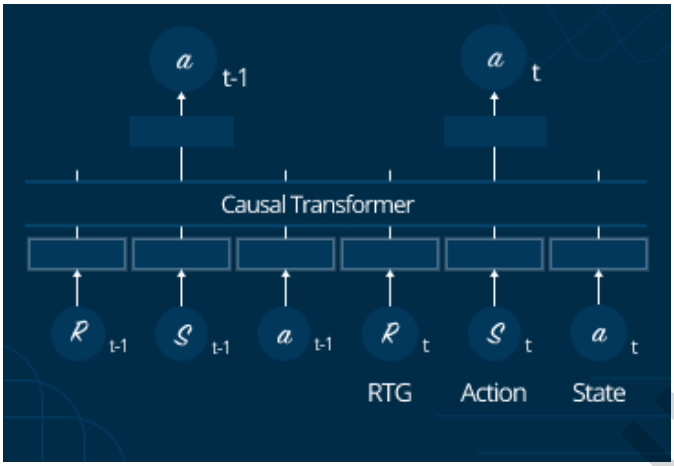


Figure 2: The abstract architecture of Decision Transformer

There also other differences as proposed in the original paper of Decision Transformer [6]. The summation/softmax over the n tokens in the standard Transformer for Language Models is replaced with the summation/softmax only the previous tokens in the sequence.

The output (Action) construction then happens using:

$$\pi(\mathbf{a}|\mathbf{s}) = (\mathrm{DT}_{\text{policy}}(\mathbf{s})), \tag{4}$$

where is the trajectory up to t timesteps(consists of previous states)

2.1.8 Offline Reinforcement Learning Evaluation

Evaluating the performance of offline RL algorithms presents unique challenges, primarily because the ultimate goal is to maximize online performance, which is not directly observable from the static dataset. When simulators are available, they provide a cost-effective means to

conduct online evaluations without real-world interaction. These simulations allow for benchmarking the performance of different algorithms by simulating the outcomes of their prescribed actions within a controlled environment.

In the absence of simulators, or as a complementary approach, Off-Policy Evaluation (OPE) [35] techniques are employed to estimate the performance of policies without requiring explicit online interaction. OPE attempts to predict the real-world effectiveness of a policy based on the data generated by a different policy. This area is notably complex and remains an active field of research, as it involves estimating what would happen under hypothetical conditions that are not represented in the data. Because this is an active research area, there several works and approaches. [10], presents a more highlighted view and benchmarks of Off-Policy Evaluation.

This presents unique challenges and necessitates the development of specialized evaluation methods. This section outlines several approaches to the evaluation of offline RL models.

2.1.8.1 Evaluating on actual environment

This involves an online interaction of the learned policy with the actual environment. So, as mentioned above this the most straightforward and most accurate and robust way to measure the performance of learned policies, by mean return, or task completion. However, this conflicts, with the general idea of offline RL, which involves learning from data, due to the fact that online interaction is costly. dangerous or not feasible at all.

2.1.8.2 Value Estimation & Related Metrics

This evaluation approach includes methods that estimate the value (V(s)) of the learned policies, via Fitted Q Evaluation (FQE), or other methods. Recall that, value of a learned policy states, how well we expect the policy to perform by following that policy on that particular state. Notably, in [25], they propose as a evaluation metric, the use of value estimation on initial states, as a way to measure the performance, of policies, denoting that higher initial values will lead to higher rewards if the corresponding policy is used.

2.1.8.3 Importance Sampling

Importance Sampling (IS), [26], is a statistical technique used to estimate the expected return of a policy under the target distribution using data collected from a different distribution. In the context of offline RL, IS can adjust the estimates of a policy's performance to account for the discrepancy between the behavior policy (which generated the data) and the target policy (which we want to evaluate). Despite its theoretical appeal, IS suffers from high variance, especially in long-horizon tasks, which can make its estimates unreliable.

2.1.8.4 Direct-Method Regression-based

Direct-Method Regression-based (DM) [17, 34] approaches are typically model-based and involve using regression techniques to estimate the value of the policy directly. The key idea is to fit a model to the observed data to predict the expected return or Q-values for given state-action pairs. The regression-based method is often biased due to the assumptions and inaccuracies in the model but tends to have significantly lower variance compared to importance sampling

methods. This makes DM particularly useful in practical applications where variance reduction is crucial for stable and reliable policy evaluation. In this work ,we focus on DM method, of estimating the policy value and use it as an evaluation metric.

2.1.8.5 Doubly-Robust

Doubly-robust (DR) [13, 7] techniques are an advanced approach to off-policy policy evaluation that combine the advantages of both importance sampling (IS) and direct-method regression-based (DM) methods. The doubly-robust estimator uses the model-based estimates from DM to reduce the variance and the reweighting from IS to correct for the bias, thus achieving a balance between the two. This combination results in more accurate and reliable policy evaluation, especially in complex environments where data distributions and policy behaviors vary significantly. DR methods have been shown to be particularly effective in various empirical studies, offering a robust solution for evaluating policies using historical data.

2.1.8.6 Model-Based Evaluation

Model-Based Evaluation involves training a model of the environment(a transition model, and a reward model) from the offline dataset and then simulating the execution of the policy within this learned model. Similar approach presented in [23], [2], and [40]. This approach can provide insights into how a policy might perform in the environment without requiring actual deployment. However, the accuracy of the evaluation is heavily dependent on the fidelity of the environment model. Misestimations of the environment dynamics can lead to incorrect assessments of a policy's effectiveness.

Transition Model

The transition model, predicts the next state of the environment given the current state and action. This model is crucial for simulating the trajectory that an agent would experience under a specific policy. Accurately modeling the transitions enables the evaluator to predict the outcomes of actions even in states or situations not covered by the offline dataset. Techniques such as deep learning and Gaussian processes are commonly used to construct transition models that can capture complex dynamics.

Reward Model

The reward model predicts the immediate reward received after taking an action in a given state. Like the transition model, the reward model is essential for simulating the outcomes of policy decisions, as it allows for the estimation of returns from a sequence of actions and states. The reward model can be learned from the offline dataset by fitting a function that maps state-action pairs to rewards, using regression techniques or deep neural networks.

By combining these models, researchers can simulate policy rollouts in a synthetic environment that mimics the behavior of the actual environment. This simulation involves iteratively applying the transition model to predict the next state and the reward model to estimate the immediate reward, given the actions chosen by the policy being evaluated. The cumulative rewards from these rollouts provide an estimate of the policy's expected return, offering insights into its potential performance in the real world.

2.1.9 Benchmarks and Datasets of Offline Reinforcement Learning

Offline reinforcement learning (RL) research often relies on benchmark environments and datasets to evaluate algorithms and compare performance. While specific benchmarks and datasets for offline RL are still emerging, several existing frameworks and datasets provide valuable resources for researchers and practitioners:

2.1.9.1 OpenAI Gym

OpenAI Gym [4] is a popular framework for benchmarking and developing RL algorithms. While primarily designed for online RL, researchers have adapted Gym environments to support offline RL by pre-generating datasets from simulated environments or replay buffers collected during online training. In addition to that, using the OpenAI Gym, everyone can populate their replay buffers from tuples generated by interacting with Gym's built-in environments.

2.1.9.2 DeepMind Control Suite

The DeepMind Control Suite [33] offers a collection of continuous control tasks suitable for offline RL research. While initially designed for online RL, researchers have utilized this suite to generate offline datasets for benchmarking algorithms in continuous action spaces.

2.1.9.3 D4RL

The most popular, offline RL related benchmark [9]. introduces a set of benchmarks specifically designed for the offline reinforcement learning (RL) setting, which reflect the complexities

encountered in real-world applications. The benchmarks focus on key properties of datasets that are critical for practical deployments. Notable aspects include:

- **Dataset Variety:** The benchmarks include datasets generated via various methods, such as hand-designed controllers, human demonstrators, and combinations of multiple policies. This variety helps ensure the benchmarks' relevance to a range of real-world scenarios.
- **Multitask Learning:** The inclusion of multitask datasets, where an agent performs different tasks within the same environment, tests the versatility and adaptability of offline RL algorithms.
- **Beyond Simple Tasks:** Moving beyond the simpler, often synthetic tasks typically used in benchmarks, these datasets are collected from environments operated by partially-trained RL agents, revealing critical deficiencies in current algorithms that may not be apparent under less challenging conditions.
- **Comprehensive Evaluation and Tools:** Along with the benchmarks, a comprehensive evaluation of existing algorithms is provided. An evaluation protocol and open-source examples are also released, supporting transparent and replicable research.
- **Community Engagement:** The release of these benchmark tasks and datasets serves as a common starting point for the community. This initiative encourages the identification of shortcomings in existing offline RL methods and fosters collaborative efforts towards advancements in this emerging field.

The datasets provided by D4RL is designed to address different aspects of offline RL challenges(such as data stiching) . These datasets include:

- **Maze2D**
- **AntMaze**
- **Gym-MuJoCo**
- **Adroit**
- **FrankaKitchen**
- **Flow**

These datasets provide a comprehensive testing ground for evaluating the robustness and efficiency of offline RL algorithms across a range of complex tasks and environments.

2.1.10 Libraries for Offline Reinforcement Learning

Several libraries and frameworks support offline RL research by providing tools for dataset management, algorithm implementation, and evaluation:

2.1.10.1 d3rlpy

The **d3rlpy** [29] library is a leading framework in the domain of reinforcement learning (RL), offering state-of-the-art, ready-to-use algorithms for both offline and online RL scenarios. It features user-friendly scikit-learn-style APIs, accessible to users at all levels of expertise. The key features of d3rlpy include:

- **Offline Reinforcement Learning:** d3rlpy is particularly effective for offline RL. It supports many offline RL algorithms and pre-collected datasets without the need for real-time data interaction.

- **Online Reinforcement Learning:** In addition to its offline capabilities, d3rlpy supports conventional online RL training algorithms without compromising on performance, enabling the solution of a broad spectrum of RL problems with a single library.

- **User-Friendly API:** Designed for ease of use, the API requires no prior knowledge of deep learning libraries, ideal for those aspiring to become RL engineers. The library is also fully documented, offering extensive support including tutorials and scripts that replicate the algorithms described in original research papers.

- **Beyond State-of-the-Art:** d3rlpy is the first RL library to integrate distributional Q functions across all algorithms. This method greatly enhances the effectiveness of Q-learning methods, elevating performance beyond traditional state-of-the-art benchmarks.

2.1.10.2 TorchRL

TorchRL [3], a recently added library, has been developed as a comprehensive control library for PyTorch, addressing the gap in native support for decision and control tasks within the framework. While this library is still in its infancy ,is designed to meet the demands of large development teams working with complex real-world data and environments, and hold very good promise. Key features of TorchRL include:

- **Generalistic Control Capabilities:** TorchRL offers a broad suite of well-integrated components that are also capable of functioning as standalone units, making it a versatile tool for a wide range of control and decision making tasks.
- **Introduction of TensorDict:** A new and flexible PyTorch primitive, TensorDict, is introduced, enhancing the development process across various branches of Reinforcement Learning (RL) and control. This primitive facilitates streamlined algorithm development and implementation.
- **Comprehensive Overview:** The library provides a detailed description of its building blocks, offering an extensive overview of its capabilities across different domains and tasks.
- **Experimental Validation:** TorchRL's reliability and flexibility have been experimentally validated, with comparative benchmarks demonstrating its computational efficiency.
- **Open Source and Community Support:** Aimed at fostering collaboration within the research community, TorchRL is publicly available on GitHub. This openness ensures greater reproducibility and facilitates long-term support.

2.1.10.3 SCOPE-RL

SCOPE-RL Library Overview

SCOPE-RL [15] is a comprehensive open-source Python software specifically tailored for offline reinforcement learning (offline RL), off-policy evaluation (OPE), and selection (OPS).. Key features of SCOPE-RL include:

- **Integration of Offline RL and OPE:** Unlike most existing libraries focused solely on policy learning or evaluation, SCOPE-RL offers a seamless integration of these two crucial aspects, enabling more flexible and comprehensive implementations.

- **Emphasis on Off-Policy Evaluation:** SCOPE-RL places a strong focus on its OPE modules, providing a variety of OPE estimators and robust evaluation protocols. This approach allows for more in-depth and reliable evaluations, enhancing the overall effectiveness of OPE compared to other available packages.

- **Risk-Return Tradeoff in OPE:** The library extends the evaluation of OPE by presenting the risk-return tradeoff in the results, which offers a more comprehensive understanding than the typical accuracy assessments found in existing literature.

- **User Accessibility:** Designed with user-friendliness in mind, SCOPE-RL features intuitive APIs, thorough documentation, and a variety of easy-to-follow examples. These resources assist researchers and practitioners in effectively implementing and experimenting with various offline RL methods and OPE estimators tailored to specific problem contexts.

Of course there are other many libraries, tools, or github repositories that provide easy integration of Offline RL processes and algorithms. But, this work, utilizes only the above tools.

Chapter 3

Methodology

The methodology of this thesis is structured around the collection and processing of replay data from the online game World of Tanks. Specifically, our focus is on a single map and game mode to ensure consistency and control over the data variables. Initially, raw replay data is collected from numerous game sessions. This data undergoes a parsing process to extract relevant features and game events. Subsequently, this parsed data is used to generate a Markov Decision Process (MDP) replay buffer. The MDP replay buffer serves as the foundational dataset for training Offline Reinforcement Learning (RL) algorithms. By structuring the data in this manner, we aim to create a robust framework for developing and evaluating offline RL models within the specific context of World of Tanks gameplay.

3.1 Data Collection

The collection of data for this research was facilitated by Wargaming, the developer of the video game. Wargaming provided the necessary data in the form of chunks of subfolders, each

corresponding to a specific replay. Within these subfolders, the data is organized as a collection of numpy arrays, each containing essential information relevant to the game, such as health, position, and other gameplay-related metrics. This section outlines the data collection process and the structure of the data provided.

The transformation of replay data from Wargaming scenarios into a format suitable for incorporation into a MDP replay buffer involves a multi-stage process. This process is essential for developing decision-making models that can operate under uncertainty, a hallmark of the MDP framework. This section outlines the methodology adopted to parse, process, and encode replays into a compatible format for MDP analysis.

3.1.1 Parsing Replay Data

The initial step involves the extraction of raw replay data from the Wargaming. This data typically encompasses a detailed log of game states, actions taken by players, and the outcomes of those actions. The outcomes can be translated into win, tie, or loss as well as some highlight events as appeared in the logged data. The parsing process is designed to identify and isolate these elements, converting them into a structured format for further analysis.

```
Input: Raw wargame replay files
Output: Structured replay data (Game states, Actions, Outcomes)
Algorithm:
1. Read raw replay files.
2. Identify player observation, player actions, and rewards.
3. Normalize data structures for consistency.
```

```
4. Output structured replay data.
```

3.1.2 Data Structure

The data provided by Wargaming was structured in a hierarchical manner. Each replay was stored in a separate subfolder, with each subfolder containing a set of numpy arrays. These arrays encapsulated various aspects of gameplay data, allowing for a comprehensive analysis of player performance. The primary numpy arrays included, but were not limited to:

Stream data (data containing information per tick/timestep about a specific feature, e.g. position)

- **Health Array:** This array stored information related to the health points of the player's character or vehicle. It allowed for tracking changes in health over the course of the game, critical for understanding player survival and combat effectiveness.It is also use the enemy health at each timestep that is used for calculating enemy health rewards in next sections.
- **Position Array:** Positional data was represented in numpy arrays, enabling the tracking of player movement throughout the replay. Positional data is crucial for spatial analysis, and also used for calculating other features and values, such as enemy distances, and the enemy base distance reward.
- **Shoot Array:** This array stored information related to the shooting events of player's character or vehicle. It allowed for tracking shoot actions, so shoot data is used to form the actions in the MDP formation.

- **Vehicle Roll, Vehicle Pitch, Vehicle Yaw , Turret Yaw, Gun Pitch Arrays:** These arrays contained information about the orientation and rotation of the player's vehicle and its turret and gun, which is important for analyzing player actions and situational awareness. It also use for calculation of other features , such as enemy targeting, direction of enemy base etc.

- **Spotted Array:** This array recorded information about the spotting of enemy vehicles, contributing to an understanding of player visibility and awareness.

- **Damage Array:** Information on damage dealt or received was stored in this array, allowing for an analysis of combat effectiveness.

Other metadata about the replay:

- **Team Information:** Information about the teams. For e.g., which team was victorious, the team each agent belongs. This information is crucial for shaping the reward for winning/ losing (or even tie).

- **Highlights:** This data is very important, as it contains events happening in the game such as killing, base capture start / complete / exit and etc. This events help give more rewards to agents.

- **Position of the bases and obstacles:** Positional data about some static objects in the map. this essentially helps calculates other resulting features derived from this data. Such as, distance from enemy base, or if there is collision with obstacles.

3.1.3 Data Preprocessing

Before conducting any analysis, the data underwent preprocessing to ensure uniformity and consistency. This included steps such as data cleaning, imputation of missing values, and standardization of data formats. The preprocessing stage was vital to guarantee the reliability of the subsequent analysis and to remove any potential data anomalies.

3.1.4 Feature Engineering

Besides the raw data including in the streams and used as features, new features are constructed by utilizing the raw data. More specifically information about the other enemy players (distance, targeting angle), the enemy base (direction, distance) and other metadata info (number of enemy tanks left, capture mode etc.)

3.1.5 Feature Extraction

Given the structured replay data, the next phase involves the extraction of features relevant to the MDP model. This typically includes observable elements of the game state, actions taken by all players, and the immediate outcomes of these actions. The MDP is defined solely from the information of this data.

The feature extraction is done by first defining the game state, or WoT state as found in the code. This state consist of all data fro the 30 players in each replay. So essentially, the initialization of this state, read all stream information as mention above (Health, Position , etc.) and creates 3 arrays that hold the information about the observations, actions and rewards for all the time-steps/ticks and all 30 players in the replay. So the shape of 3 arrays is (T, A, K), where

T is the number of ticks in the replay, A is number of players (30) and K the dimension of each component (states, actions, rewards). Essentially these 3 arrays, are used to retrieve the states, actions and rewards of each of the 30 players. So , from each replay, 30 episodes are generated each containing the states, actions and rewards for each of the 30 players.

3.1.6 Feature Scaling

In reinforcement learning (RL), feature scaling is a crucial preprocessing step that can significantly influence the performance of learning algorithms. Proper scaling ensures that the state representation remains consistent across different environments, aiding in the convergence of learning algorithms and improving their efficiency. This subsection outlines the scaling methods applied to the features in this work: Min-Max scaling and centering around zero.

3.1.6.1 Min-Max Scaling

Min-Max scaling is a normalization technique that adjusts the features to ensure they range between 0 and 1. This is particularly useful in RL where different features may have varying scales and ranges, potentially leading to biases in the learning process. The transformation is defined as follows:

$$\text{Scaled Value} = \frac{\text{Observation} - \text{Min Value}}{\text{Max Value} - \text{Min Value}} \tag{5}$$

where ”Min Value” and ”Max Value” are the minimum and maximum values observed for that feature across the dataset. This scaling method is straightforward and ensures that all features contribute equally to the learning process, making it easier for the algorithm to learn.

Below is the overall procedure of generating the MDP data from the structured replay data:

```
Input: Structured replay data
Output: Feature vectors representing states, actions, and outcomes.
Algorithm:
1. For each game state in the replay data, extract  features.
2. Encode actions and outcomes as vectors.
3. Calculate rewards
4. Output observations, actions, rewads as 3-D arrays of shape
(# of ticks, # of agents, dimension of states/actions/reward).
```

3.1.7 Encoding for MDP Replay Buffer

The final step in preparing the data for the MDP replay buffer involves encoding the extracted features into a format that is directly usable by MDP models. This includes structuring the data into tuples of (state, action, reward), which are essential for training reinforcement learning algorithms under the MDP framework. The next state and done flags, required, are calculated during the training

```
Input: Feature vectors from previous step
Output: MDP-compatible replay buffer data
Algorithm:
1. Initialize an empty replay buffer.
2. For each set of feature vectors, create a tuple (s, a, r, s', done).
3. Populate the replay buffer with these tuples.
```

4. Ensure compatibility with MDP model training routines.

5. Output the populated MDP replay buffer.

More specifically a tuple (s, a, r) holds the below information

3.1.7.1 State

Observations		
Feature	**Description**	**Dim**
Health	Health of the agent at t	1
Position	The coordinates of the vehicle	3
Enemy Distance	distance from enemy agents	15
Pitch	Records information on the pitch of the vehicle	1
Roll	Records information on the roll of the vehicle	1
Turret Yaw	Records information on the yaw of the turret	1
Gun Pitch	Records information on the pitch of the gun	1
Yaw	Records information on the yaw of the vehicle	1
Enemy Targeting	relative angle between agent and enemy agents indicating targeting or not	15
Spotted	Logs information on spotting enemy vehicles	1
Damage	Stores information on damage received and the part of the vehicle that reaceive the damage	2
Enemy Tanks Count	Logs information on remaining number of enemy tanks	1
Enemy Base Direction	Records information about the agent direction to enemy base (x, y)	2
Object Collision	A boolena value indicating if there is intersection between agent position and one of the obstacle objects	1

Table 1: State of the MDP

3.1.7.2 Actions

In this Markov Decision Process (MDP) framework, the agent can perform a set of deterministic continuous actions. Note that some actions cannot be performed together. For e.g. move right vs move left, gun up vs gun down, etc. And these actions are not exist in the dataset. The table below 2 summarizes these actions and their corresponding descriptions:

Actions	
Action	**Description**
Move Right	makes agent move right
Move Left	makes agent move left
Move Backward	makes agent move backward
Move Forward	makes agent move forward
Turret Right	makes agent rotate the tank's turret to the right
Turret Left	makes agent rotate the tank's turret to the left
Gun Pitch Up	makes agent rotate the tank's gun upside
Gun Pitch Down	makes agent rotate the tank's gun downside
Shoot	makes agent shoot

Table 2: Deterministic Continuous Actions of the MDP

3.1.7.3 Rewards

In reward modeling, the agent receives rewards based on its actions and interactions with the environment. These rewards will be used to reward agents during the training. These rewards can include:

- **Sparse Rewards:** At the conclusion of each match, the agent receives a win reward if its team emerges victorious, a tie reward if the match ends in a draw, and a loss reward if its team is defeated. These rewards incentives the agent to contribute to its team's success and penalize undesirable outcomes. Sparse rewards also includes Highlight rewards.
- **Immediate Rewards:** Rewards received by the agent immediately after taking an action in a given state. In the context of World Of Tanks, immediate rewards, are rewards given from being close to opponent's base or reward for reducing enemy total health. Note that these reward will be used for evaluation (see next section), and for training of the Decision Transformer (since for its evaluation, it needs the previous collected rewards). For other algorithms (IQL, CQL, DDPG), all the other rewards are used, and not the immediate rewards.
- **Return-to-go:** The cumulative sum of rewards obtained from a specific timestep t over a sequence of actions and states until the end or a terminal T. This is only applicable, in the Decision Transformer algorithm.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- **Highlight Rewards:** Various events during game play trigger rewards or penalties for the agent. For instance, successfully eliminating an opponent (kill) results in a positive reward, while being eliminated by an opponent incurs a negative reward. Additionally, events related to capturing objectives, such as capturing a base, beginning/ending a capture process, or completing a capture, can lead to rewards or penalties depending on the strategic significance of the action.

the values of these rewards are given below:

- **Win Reward:** *100*
- **Tie Reward:** *0*
- **Loss Reward:** *-100*
- **Kill Reward:** *50*
- **Capture Begin Reward:** *20*
- **Capture Complete** *75*
- **Capture Exit:** *-20*
- **Enemy base distance Reward:** *1- Euclidean-distance*$(pos_{agent} - pos_{enemy-base})$
- **Enemy Health Reward:** *1- normalized-enemy-health-remaining*

Each of these rewards plays a crucial role in shaping the agent's behavior and guiding its learning process in reinforcement learning tasks. It is noted, that all the rewards(except the enemy base

and health which are already normalized when calculated) are scaled by multipling it with 0.01, so are in the [-1, 1] range.

3.2 Modeling

In this section, we delve into the modeling of Offline Reinforcement Learning (RL) algorithms, which are designed to learn optimal policies from a fixed dataset without further interaction with the environment.

To address this, we use or modify several state-of-the-art implementations of offline RL algorithms supported by both d3rlpy and TorchRL.These includes Implicit Q-learning (IQL), Conservative Q-Learning (CQL), Decision Transformer (DT), DDPG and Behavioural Cloning. These algorithms incorporate various strategies to mitigate the issues of overestimation and distributional shift. CQL penalizes the Q-values of out-of-distribution actions to prevent the learned policy from exploiting these actions. IQL queries and updates Q values only in actions seen in the Dataset. DDPG, is used to see what is the effect of including algorithms that do not account for the offline RL problems, such as overestimation. Finally, DT offers a different paradigm in Offline RL, and does not suffer from overestimation of value, since it has different architecture, however it does not have the property of stiching different good behaviours from dataset together.

The training process involves the iterative optimization of the policy and value functions using the MDP replay buffer generated from the parsed game data. We use standard RL techniques such as Q-learning and policy gradients, adapted to the offline setting. The performance of these algorithms is evaluated based on their ability to achieve high rewards and generalize

to new game scenarios within the constraints of the offline dataset. By comparing the results across different algorithms, we aim to identify the most effective approaches for offline RL in the context of World of Tanks gameplay.

The hyperparameter details of each algorithm can be found at the end of this document.

3.3 Evaluation

This sections covers the evaluation of the learned policies.

3.3.1 Policy Value Estimation

The Direct Method (DM) is a model-based approach that uses the initial state value estimated by Fitted Q Evaluation (FQE)[17]. It first learns the Q-function from the logged data via temporal-difference (TD) learning and then utilizes the estimated Q-function for OPE as follows:

$$\hat{J}_{DM}(\pi; \mathcal{D}) := \frac{1}{n} \sum_{i=1}^{n} \sum_{a \in \mathcal{A}} \pi(a|s_0^{(i)}) \hat{Q}(s_0^{(i)}, a) = \frac{1}{n} \sum_{i=1}^{n} \hat{V}(s_0^{(i)}),$$

where $\hat{Q}(s_t, a_t)$ is an estimated state-action value and $\hat{V}(s_t)$ is the estimated state value. DM has lower variance compared to other estimators but can produce large bias caused by approximation errors of the Q-function.

The key advantage of FQE is that it produces a Q function that can estimate evaluation metrics more accurately than the Q function learned during the training phase. By focusing on the evaluation aspect, FQE enhances the reliability of the policy performance assessment, ensuring that the metrics reflect the true quality of the policy more precisely.

Essentialy, it outputs the initial policy value estimation meaning that the higher the initial value the more rewards the policy will return when deployed to the initial states.

3.3.2 Evaluation using Model-Based Dynamics

In reinforcement learning, evaluating the performance of a learned policy is crucial for assessing its effectiveness. One approach to evaluating a policy is through model-based dynamics of the environment, where deep neural networks are used to predict the next state transition and reward based on the current state and action.

3.3.2.1 Designing a Reward Function for Building a Dynamics Model

In the development of reinforcement learning systems, particularly those involving complex environments like games similar to World Of Tanks with strategic objectives (e.g., warfare scenarios involving tanks), it is crucial to distinguish between the reward functions used for training policies and those used for modeling dynamics, which include both transition and reward models. The need for this distinction often arises from the inadequacies of conventional reward structures, especially when dealing with sparse rewards, which are typical in environments where significant events (like destroying an enemy tank) are rare, in the an horizon of a game.

Sparse rewards pose a significant challenge in training robust neural networks because they provide limited feedback signals for learning. To address this issue, we introduce a continuous reward function inspired by the idea of Reward-consisted Dynamics in [20] specifically designed

for the dynamics model, which aims to more effectively capture the nuances of policy performance through a blend of immediate rewards. This tailored reward function incorporates two primary components:

- **Remaining Health of Enemy Tanks:** The reward increases as the remaining health of enemy tanks decreases. Mathematically, this can be expressed as $R_{health} = 1 -$ remaining health of enemy tanks, where a lower health value results in a higher reward.
- **Distance from Enemy Base:** The closer the agent is to the enemy base, the higher the reward. This is quantified as $R_{distance} = 1 -$ normalized distance from enemy base, promoting aggressive forward movement and positioning.

Penalties for Simultaneous Incompatible Actions

In scenarios such as tank simulations, certain actions are mechanically or strategically incompatible, such as moving forward and backward simultaneously. To discourage the selection of these conflicting actions, a penalty is imposed whenever they are executed concurrently. The penalty can be mathematically represented as follows:

$$R_{penalty} = \begin{cases} -1 & \text{if actions that cannot be performed simultaneously} \\ & \text{(e.g., forward/backward, gun up/gun down)} \\ 0 & \text{otherwise} \end{cases}$$

This penalty is subtracted from the total reward for the time step, thereby reducing the overall reward obtained when these incompatible actions are taken.

Penalty for Inaction

While the model allows for no action as a valid choice, strategically, continual inaction (when other actions are possible and could yield a better strategic position) is undesirable. This is particularly true in dynamic environments where proactive measures are necessary to achieve objectives. Therefore, a penalty is also applied for inaction, to promote engagement and decision-making in the simulation:

$$R_{inaction} = \begin{cases} -1 & \text{if there is no action at all} \\ 0 & \text{otherwise} \end{cases}$$

The combination of these factors into a single continuous reward function R_t is formulated as follows:

$$R_t = \alpha R_{health} + \beta R_{distance} + R_{penalty} + R_{inaction}$$

The combination of these factors into a single continuous reward function R_t is formulated as follows:

where α and β are weighting factors that balance the importance of health reduction versus positional advancement. The values of α and β are ranged between 0 and 1 and together they add up to one. This design facilitates the generation of a more informative and consistent training signal for the dynamics model, thereby enhancing the model's ability to predict and

evaluate the effectiveness of various policies in achieving strategic objectives. Ultimately, the design of such a continuous reward function aims to build a more robust neural network model capable of accurately simulating and predicting complex interactions in dynamic and strategic environments.

3.3.2.2 Model Architecture

Two separate deep neural networks are trained to model the dynamics of the environment:

1. **Next State Prediction Model (NSPM):** This model takes as input the current state s and action a and predicts the next state s'.

2. **Reward Prediction Model (RPM:** This model takes as input the current state s and action a and predicts the reward r received from transitioning from state s to s'.

Both models are trained using backpropagation and optimization algorithms such as stochastic gradient descent (SGD) or Adam on historical data collected during the agent's interaction with the environment.

Model	**Parameters**	**Type**
Next State Prediction Model (NSPM)	Layers: 2 Units per layer: 128 Hidden Layer Activation function: ReLU Activation function: Linear	Fully Connected
Reward Prediction Model (RPM)	Layers: 2 Units per layer: 128 Hidden Layer Activation function: ReLU Activation function: Sigmoid	Fully Connected

Table 3: Parameters details about the two models that define the dynamics model, the transition model, and the reward model

Both models are trained on 10M (s,a,s',r) training data tuples that derived from the replays. And evaluated on 400K testing tuples.

3.3.2.3 Evaluation using Mean Squared Error (MSE)

To evaluate the performance of the NSPM and RPM, Mean Squared Error (MSE) is computed on a validation dataset. The validation dataset consists of state-action pairs (s, a) and their corresponding next states s' and rewards r, which are not seen during training.

The MSE for the NSPM is calculated as:

$$\text{MSE}_{\text{NSPM}} = \frac{1}{N}\sum_{i=1}^{N} \|s'_i - \hat{s}'_i\|^2$$

where N is the number of samples in the validation dataset, s'_i is the true next state, and $\hat{s}'_i$ is the predicted next state by the NSPM.

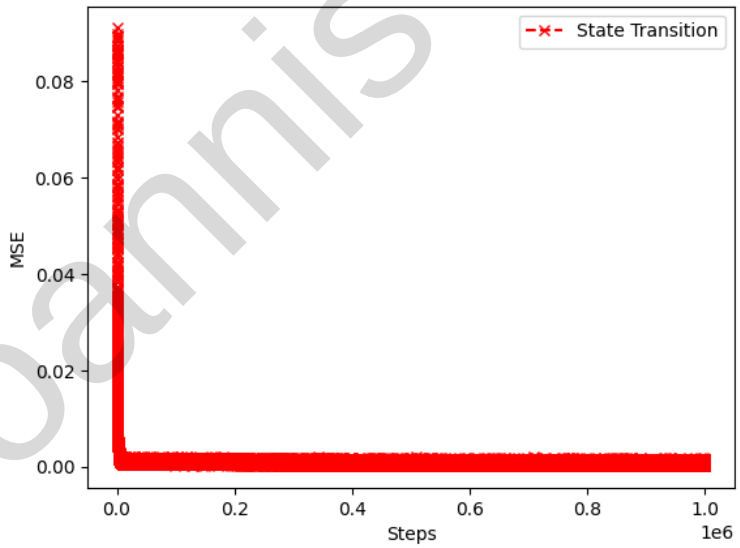
Similarly, the MSE for the RPM is computed as:

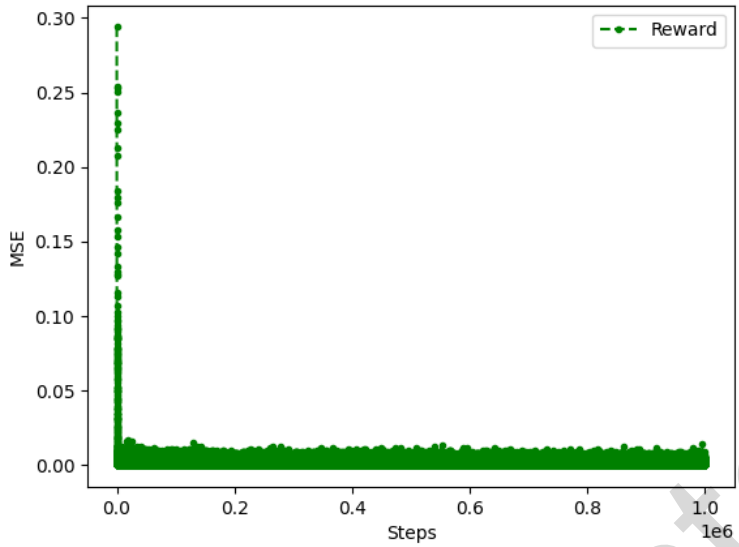
$$\text{MSE}_{\text{RPM}} = \frac{1}{N}\sum_{i=1}^{N} \|r_i - \hat{r}_i\|^2$$

where r_i is the true reward, and $\hat{r}_i$ is the predicted reward by the RPM.

A lower MSE indicates better accuracy of the models in predicting the next state and reward, respectively. These metrics provide insights into the performance and generalization capabilities of the NSPM and RPM in modeling the dynamics of the environment.'

Below are the losses of the above models, indicating a resonanbly low values,





3.3.2.4 Offline Policy Evaluation Procedure

To evaluate the learned policy, the following procedure is followed:

1. **Rollout:** The evaluation starts with an initial state s_0 obtained from the test dataset. The policy is then executed in the environment, generating a sequence of actions according to the policy, for k steps.

2. **Model Prediction:** Using the next state prediction model, the next state s' is predicted for each state-action pair encountered during the rollout. Similarly, the reward prediction model is used to predict the reward r received at each state-action pair.

3. **Policy Evaluation:** The performance of the policy is evaluated based on the predicted next states and rewards. More specifically, mean return of rollouts on the initial states of

the test dataset are computed and compared against a baseline(return on actions taken in the states in the test dataset).

3.3.2.5 Baselines

In addition to the above, we predict the reward for the state - actions pairs taken in the evaluation dataset, of high rewards episodes, to see how they perform using the above criteria and reward function, and compare them with the learned policies. And we also, use the Behaviour Cloning policy as well.

Baseline 1: Behaviour Cloning (BC) Policy

- The Behaviour Cloning (BC) policy is trained on the entire training dataset.
- Evaluation is conducted through simulated rollouts in the dynamics model.
- The rollouts are done by using both the next transition and reward model.

Baseline 2: High Reward Episodes

- High reward episodes are defined as episodes falling into the 66th percentile (total reward ≥ 0.5 normalized) of evaluation dataset.
- Only the reward model is used to predict the returns for these episodes, as the transitions are already available in the dataset.

3.3.2.6 Challenges

However, model-based evaluation also poses some challenges and limitations:

- **Model Accuracy:** The accuracy of the learned models directly affects the reliability of policy evaluation. Especially, in the World Of Tanks scenarios, where the episodc steps are long, error accumulates, making it more prone inaccurate results. Inaccurate models may lead to biased performance estimates.

- **Reward :** The reward function used for this model, does not contain the actual and more representative rewards in the World of Tanks context. Since originally, the nature and the data of the replays (in the World of Tanks context, include more represantive rewards such as win/loss, capture/kill rewards, which are sparse rewards difficult to be predicted by a model. However, using the above continuous and consistent over time reward function we can get an indicating performance of the policies.

Overall, model-based dynamics provide a powerful framework for evaluating learned policies in reinforcement learning, offering both efficiency and interpretability in assessing policy performance.

Chapter 4

Experiments

In this section, we present the details of the experiments conducted to evaluate the performance of Offline Reinforcement Learning (RL) algorithms in the context of World of Tanks. The primary objectives of the experiments were to assess the impact of training data size on algorithm performance and to compare the performance of different RL algorithms on the same dataset.

4.1 Libraries Used for Training Offline RL in World of Tanks

The training of Offline Reinforcement Learning (RL) models for the World of Tanks game was facilitated using a variety of specialized libraries. These libraries provided essential tools for data processing, model implementation, and evaluation. Below is a detailed description of the key libraries employed:

- **NumPy** [11]: In our experiments, NumPy was specifically used for processing game replays and converting them into a Markov Decision Process (MDP) replay buffer, as well as some other supplemnentary functions.

- **d3rlpy**: . It was primarily used for saving the replay buffers into a compressed .h5 file format, which is efficient for handling large replay files. Additionally, d3rlpy was utilized for training the Decision Transformer algorithm, and Behaviour Cloning.

- **TorchRL** : In our experiments, TorchRL was used for training several algorithms, including Conservative Q-Learning (CQL), Implicit Q-Learning (IQL), and Deep Deterministic Policy Gradient (DDPG).

- **Matplotlib** [12]: It was used for visualizing the training progress and performance metrics of our RL models, including plotting the different results obtained from various training sessions and algorithms.

- **SCOPE-RL** In our experiments, SCOPE-RL was utilized to evaluate the performance of different policies trained via Offline RL, using Policy Value Estimation.

The combination of these libraries provided a robust and flexible framework for training Offline RL models in the context of the World of Tanks game, enabling effective data processing, model training, and performance evaluation.

4.2 Algorithms

The algorithms used in the experiments are the ones already mention in chapter 1 and related work. These algorithms are :

1. **Conservative Q-Learning**
2. **Implicit Q-Learning**
3. **Behavioural Cloning**
4. **Decision Transformer**
5. **DDPG**

4.3 List of Experiments Conducted

This section details the specific experiments conducted to evaluate the efficacy of offline reinforcement learning (RL) algorithms in the World of Tanks game environment. Each experiment is designed to explore the effects of varying key parameters—discount factor (γ), Distibution Shift effect , and dataset size—on the performance of Conservative Q-Learning (CQL), Implicit Q-Learning (IQL), and Decision Transformer algorithms.

4.3.1 Experiment 1: Impact of Discount Factor (γ)

Objective: To assess how the choice of γ influences the algorithms' prioritization of immediate versus future rewards.

Parameters:
- Discount Factor (γ): 0.99, 0.995, 0.9995, 0.9999
- Dataset Size: 178k episodes

Algorithms Tested: CQL, IQL

4.3.2 Experiment 2: Distribution Shift effect

Objective: Comparison of standard RL algorithm (DDPG) vs offline RL algorithms(IQL, CQL) that handle overestimation and OOD(Out-Of-Distribution) actions.

Hypothesis: Since DDPG does not account (Out-Of-Distribution) actions, we expect that DDPG will overestimate actions that are not seen in the dataset. And, while we have a huge dataset, recall that there still out-of-distribution actions, since the way we model actions allow the presence of move forward and backward actions together. Whilst, in data this is not the care since these actions can be taken by player and thus not recored in the data.

Parameters:
- Discount Factor (γ): 0.9999
- Dataset Size: 178K episodes

Algorithms Tested: DDPG vs CQL

4.3.3 Experiment 3: Dataset Size Comparison

Objective: To explore how the volume of training data affects the algorithms' ability to learn effective policies.

Parameters:
- Discount Factor (γ): 0.9999
- Dataset Size: 600, 178k episodes

Algorithms Tested: CQL, IQL

Each experiment is systematically designed to isolate the effects of one or more variables, providing insights into the optimal deployment of offline RL algorithms in complex game environments. The results from these experiments are expected to contribute significantly to the field of reinforcement learning, particularly in applications where live interaction with the environment is constrained.

4.3.4 Evaluation Metrics

Since there is no access to environment to retrieve actual returns of the policies, the evaluation metrics that are gonna be used fro the evaluation of the learned policies are:

- Policy Initial Value Estimation (V(s0))
- Return from Model-Based Environment Rollouts (R-MDP)

4.4 Results and Analysis

This section discusses the evaluation results of different reinforcement learning algorithms including Deep Deterministic Policy Gradient (DDPG), Implicit Q-Learning (IQL), Conservative Q-Learning (CQL), and Decision Transformer (DT). We compare the policy value estimations and mean returns of these algorithms through visual analyses.

4.4.1 Policy Value Estimation

The evaluation of the learned policies was conducted using estimated policy value estimation. However, this approach is only applicable to value-based approaches (CQL, IQL, DDPG), and it is important to note that this method is not the primary source of evaluation due to the

tendency of offline reinforcement learning to suffer from policy value overestimation. It can still be used to indicate the performance, but having it also check if corresponds to other evaluation metrics like the return of the environments.

This evaluation is mainly used to highlight this issue.

4.4.2 Mean Return Comparison

Similarly, the mean returns for each algorithm are illustrated in Figure 6. These returns are indicative of the overall performance of the algorithms and their ability to maximize rewards.

Each of these plots provides a comprehensive overview of the algorithms' performances and are essential for making informed decisions regarding the selection and further development of reinforcement learning strategies.

LIST OF TABLES

E1				
Policy Value Estimation				
Algorithms	γ-0.99	γ-0.997	γ-0.9995	γ-0.9999
CQL	-400	-0.005	0.12	0.18
IQL	0.03	0.05	0.08	0.15
DDPG	-	-	-	**3.5**

Table 4: Estimated Policy (Initial) Value Estimation of the model-based algorithms for Various Discount Factors. For DDPG, only one experiment is done just to see the data distribution shift effect.

Experiment 1 and 2				
Estimated Mean Return				
Algorithms	γ-0.99	γ-0.997	γ-0.9995	γ-0.9999
CQL	19	86	126.0	**151.0**
IQL	26	60	97	110.0
DDPG	-	-	-	-120

Table 5: Estimated Mean Return of Model-based Algorithms for Various Discount Factors. For DDPG, only one experiment is done just to see the data distribution shift effect.

Experiment 1 and 2	
Estimated Mean Return	
Algorithms	γ-0.9999
BC	79.85
CQL	**151.0**
IQL	110.0
DT	126
DDPG	-120

Table 6: Estimated Mean Return of Different Algorithms at gamma 0.9999 .

Experiment 3	
Estimated Mean Return(CQL)	
Data Size	**g-0.9999**
600	47
178K	151

Table 7: Estimated Mean Return of CQL in two different data sizes. The difference showcases importance of data in Offline RL

4.4.2.1 Gamma Effect

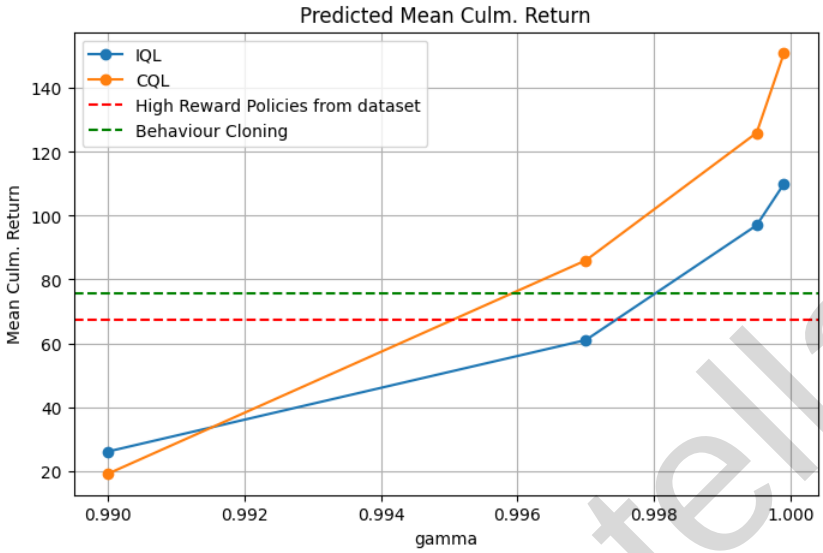


Figure 3: Estimated mean return plot of IQL and CQL at different gamma values. In addition , estimated mean return of baselines is plotted (Behavioural Cloning and high reward policies.

The plot illustrates the impact of the discount factor, gamma, on the predicted mean cumulative return for two Offline Reinforcement Learning (RL) algorithms: Implicit Q-Learning (IQL) and Conservative Q-Learning (CQL). As gamma increases from 0.990 to 1.000, both algorithms exhibit an upward trend in their mean cumulative returns. Initially, for gamma values close to 0.990, both algorithms perform below the high reward policies from the dataset, represented by the red dashed line, and behaviour cloning algorithm represented by green line. This red dashed line signifies the return on high reward episodes within the dataset. However, as gamma approaches 1.000, the performance of both algorithms improves significantly. Green line, showcases the return of the BC algorithm of its rollout in the dynamics model. Notably, CQL demonstrates a more pronounced increase in returns compared to IQL, especially at gamma

values closer to 1.000, where it surpasses IQL and achieves a higher mean cumulative return. This phenomenon can be attributed to the extended time horizon in the replays, which spans around 800 timesteps, and not only that most of the rewards(win/loss, and highlights rewards) happen much later to the game. With a higher gamma, which places more emphasis on future rewards, the algorithms are better able to exploit the longer-term benefits and also able to see the rewards from this long-term events such as kill rewards.

Another effect of gamma can be seen, also in the probability distribution of actions, of the learnt policies. Below we can see the action probability distribution in the rollouts of CQL trained with gamma at 0.99 and then with gamma at 0.9999.

Note that negative returns happens, because there many noop or contradicting actions at the same time (e.g move forward and move backward together).

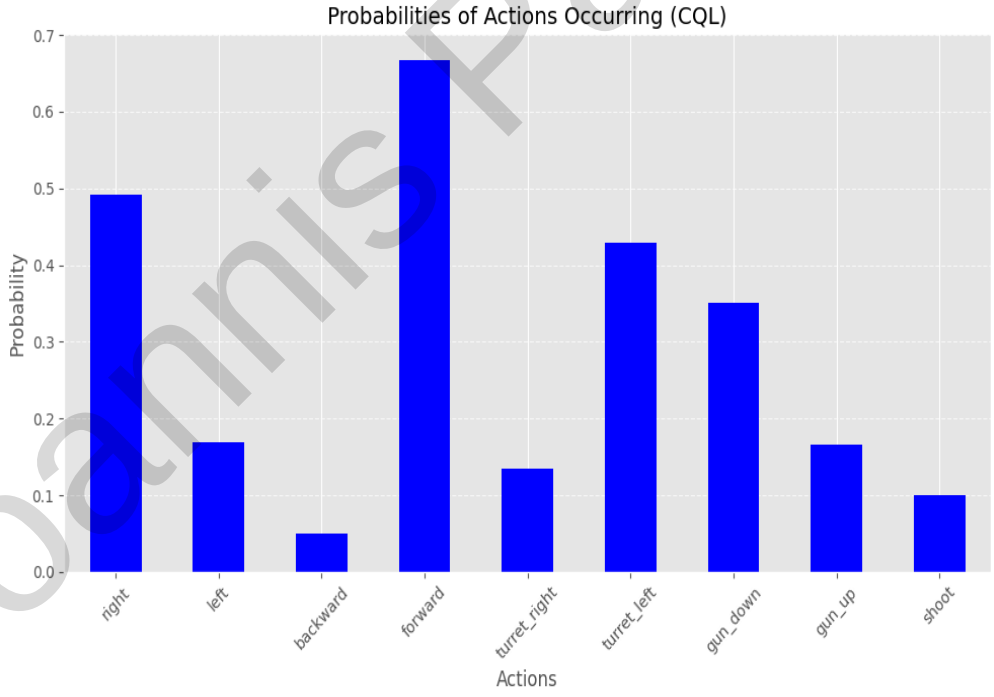


Figure 4: CQL percentage of actions taken of actions at gamma 0.99

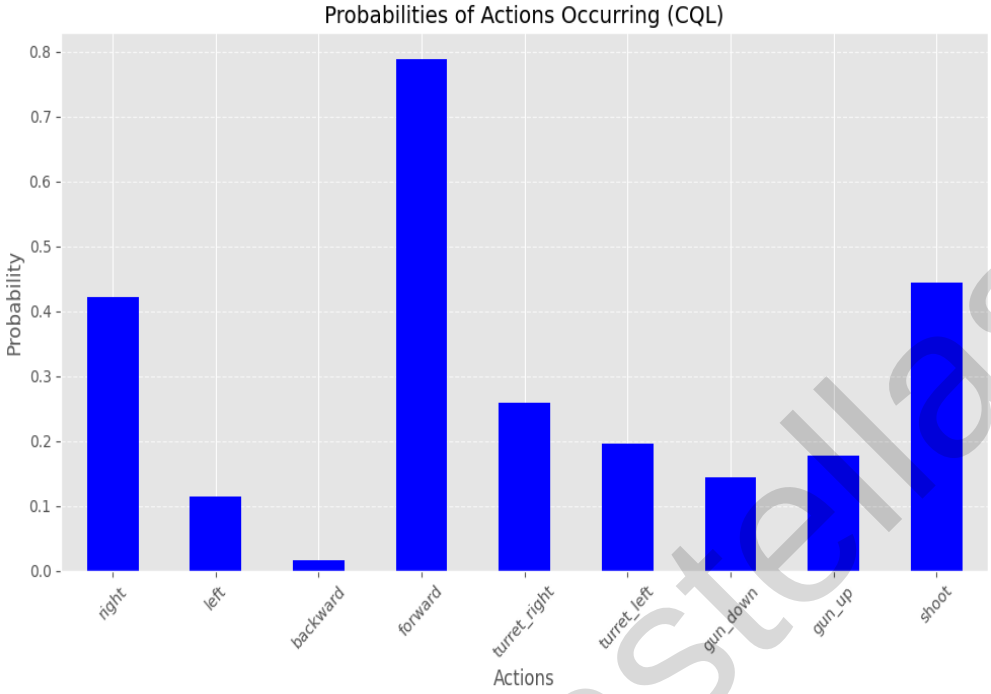


Figure 5: CQL percentage of actions taken at gamma 0.9999

Chapter 5

Discussion

5.1 Results

5.1.1 Evaluating Offline Reinforcement Learning Properties in Experiments

The experiments conducted as part of this thesis were meticulously designed to evaluate several core properties of offline reinforcement learning. Offline RL, which involves training a policy entirely from a fixed dataset without additional online interactions with the environment, poses unique challenges and opportunities. Key aspects examined in these experiments include the improvement of policy behavior over the baseline behaviors(high rewards episodes, BC) present in the dataset, the impact of data distribution shifts, and the performance divergence of derived policies from the original data-collecting policies.

Improvement Over Dataset Behaviors

One of the primary objectives was to assess whether the trained policies could surpass the performance of the behavior policies that generated the training data. By leveraging advanced

offline RL algorithms capable of effectively learning from offline data and also mitigating several challenges, mainly, data distribution shift.The experiments demonstrated noticeable improvements in performance within the simulated environment using the dynamics model, with a specific simpified reward function. These improvements were quantified by comparing the mean return of simulated rollouts, against those exhibited by the baselines. More specifically, all offline RL algorithms (CQL, IQL, DT) acquiring mean return of 151, 110, 126 respectively, outperformed both BC and high rewarded policies, of mean returns of 79.85 and 68.

Handling Data Distribution Shifts

Another significant aspect of the investigation focused on the robustness of policies when faced with scenarios not well-represented in the training data. Offline RL inherently suffers from issues related to data distribution shifts, where the policy encounters state-action pairs that are underrepresented or absent in the training dataset. Through the application of various regularization techniques(CQL) and modified architectures(IQL), the experiments aimed to mitigate these effects, enhancing the policies' generalization capabilities, and from the results we can see that, based on the returns but also the value estimations, IQL and CQL mitigate this problem, while DDPG suffers extremely. DDPG has quite large initial policy value (3.5) compared to the other two. However, it has return of -120, mainly because it perform actions simultaneously that are forbidden, and thus penalized.

Decision Transformer, Behaviour Cloning, are prone to this problem, since they do not rely on value functions, and that's why they don't perform action together that are forbidden.

Policy Performance Divergence

Finally, the experiments explored how the derived policies performed under conditions that differed from those of the training data. And that in general, the learned policies, behave differently, than the dataset. This exploration is crucial in offline RL, as it indicates the policy's ability to generalize beyond the confines of its training environment and the data, and that hopefully will learn to stich different good behaviours and trajectories together. The findings revealed that while some policies adapted well and maintained high performance levels, others exhibited a drop in effectiveness, underscoring the importance of careful algorithm selection and tuning to address the challenges posed by offline RL.

This is highlighted by the plots below, showcasing the action probabilities of the dataset and the learned policies.

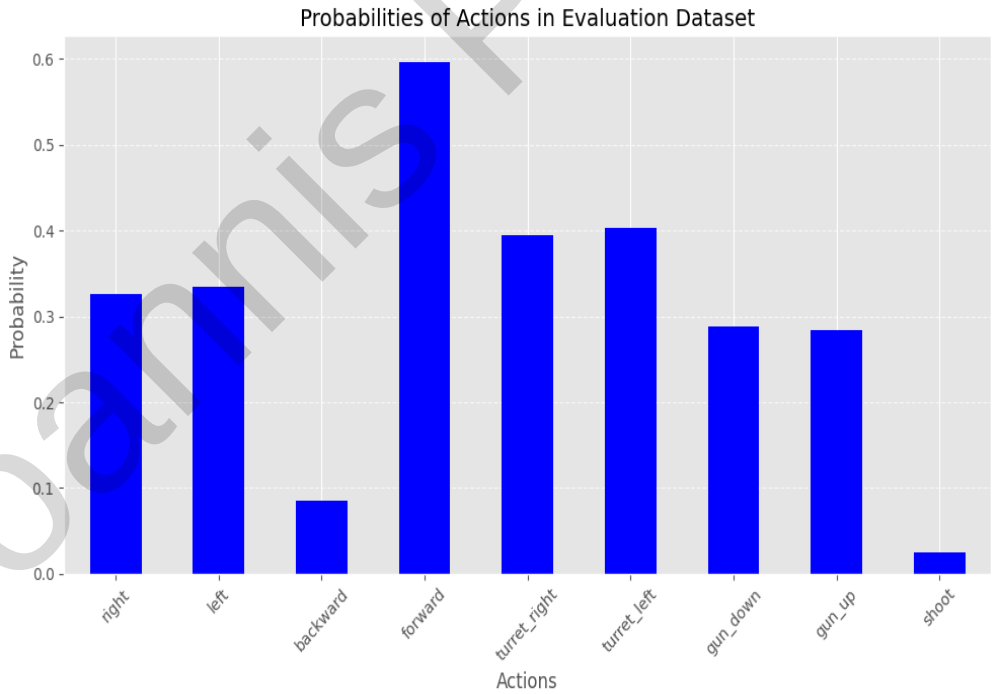


Figure 6: Percentage of actions taken in Dataset

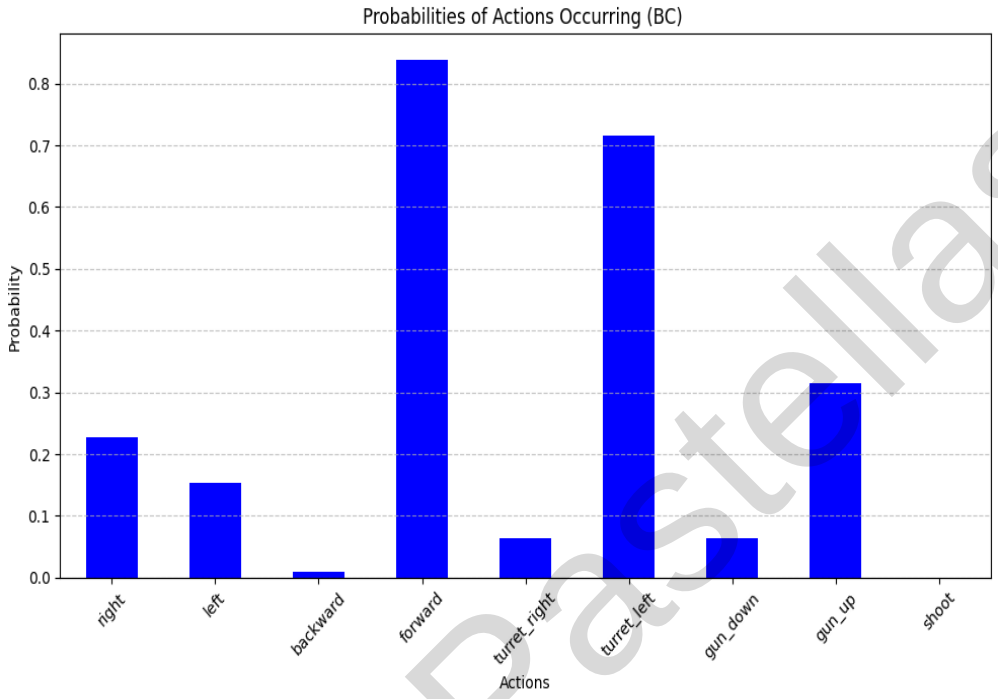


Figure 7: Percentage of actions taken by Behaviour Cloning

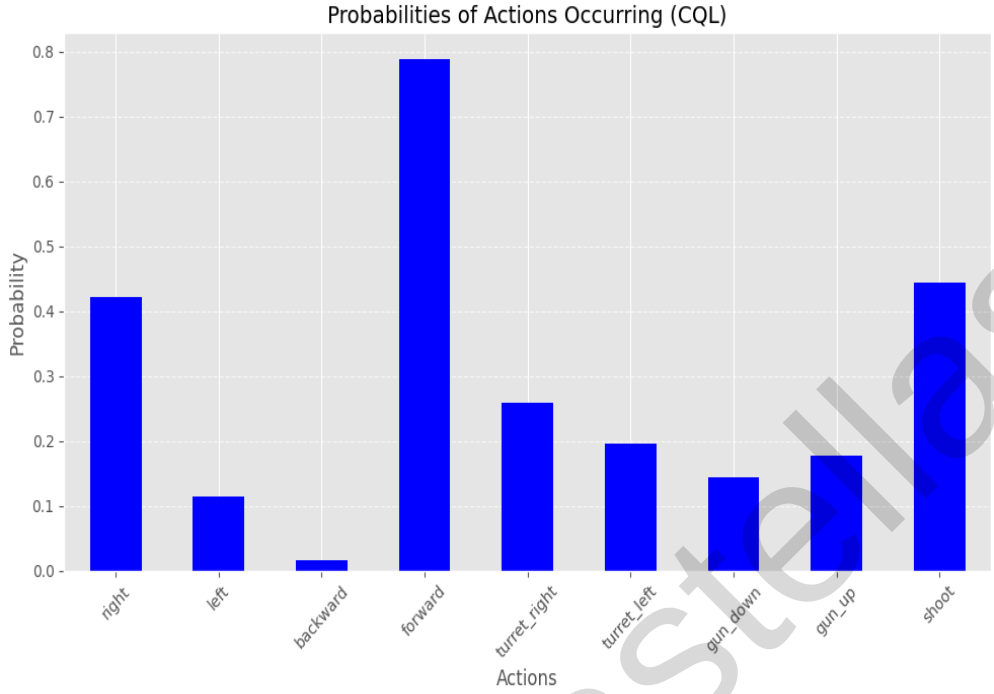


Figure 8: Percentage of actions taken by Conservative Q-Learning

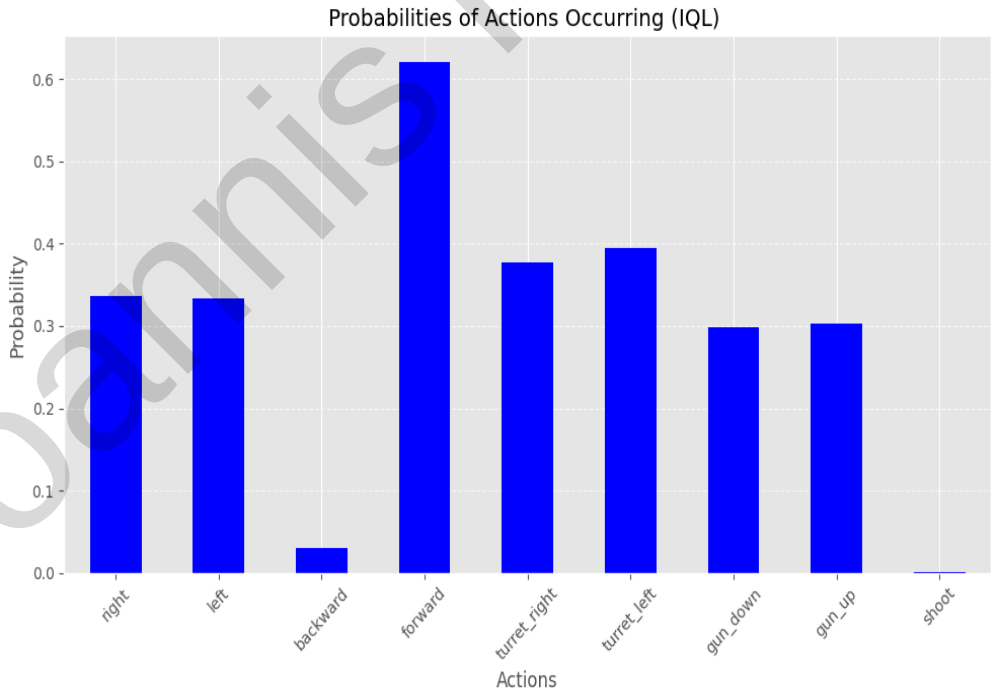


Figure 9: Percentage of actions taken by Implicit Q-Learning

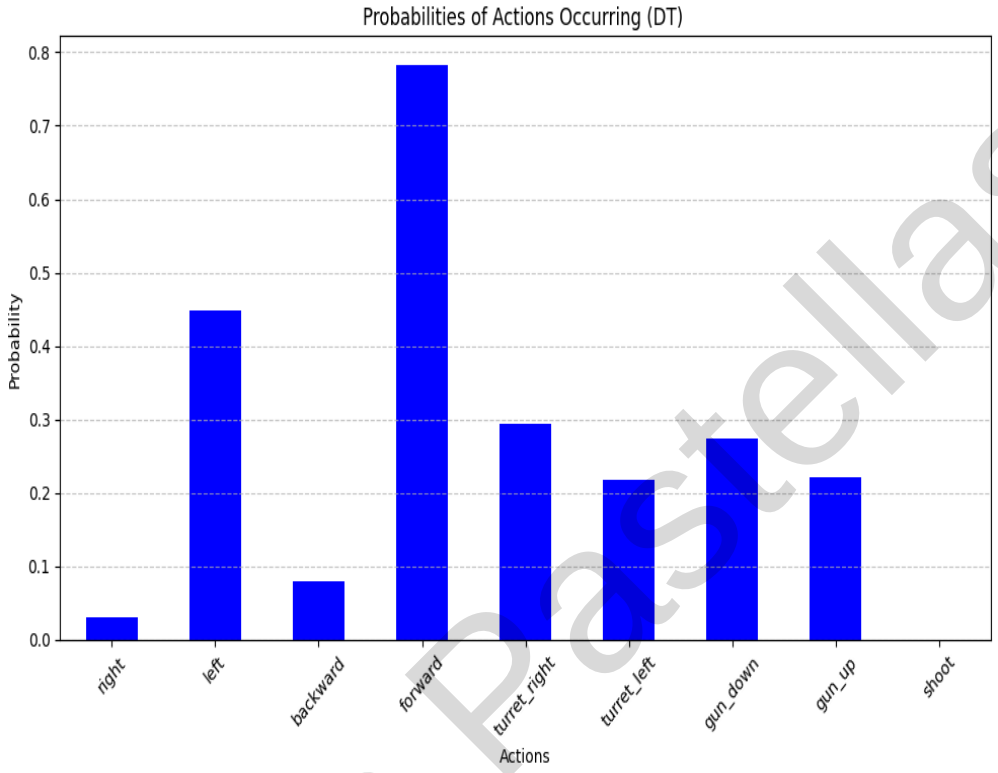


Figure 10: Percentage of actions taken by DT

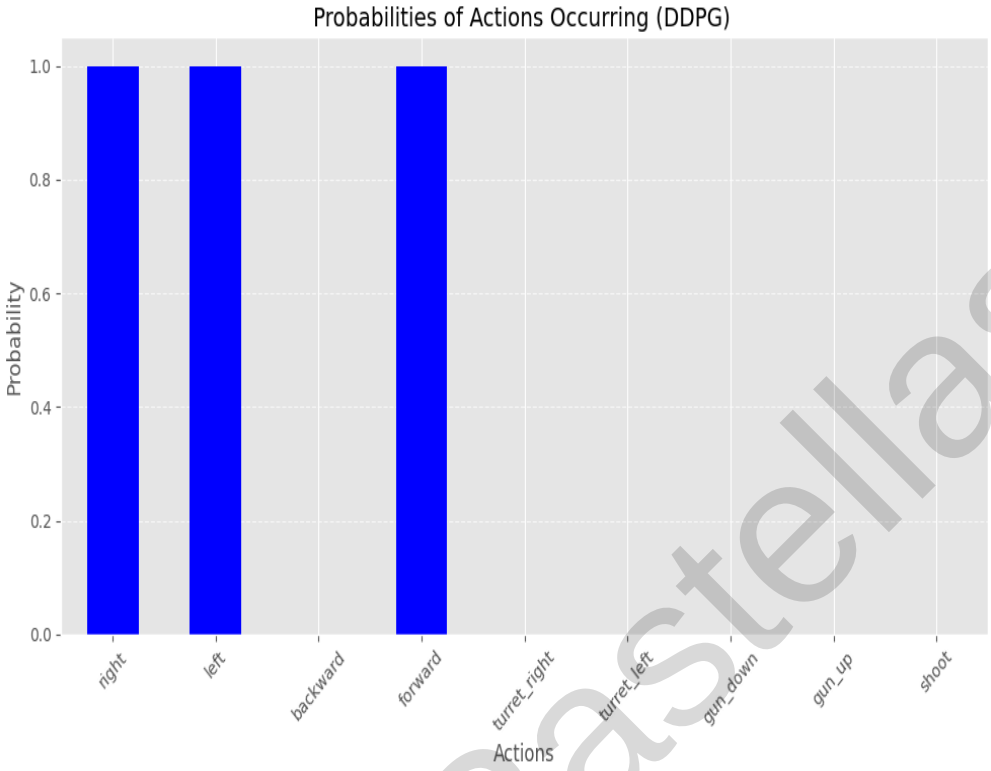


Figure 11: Percentage of actions taken by DDPG

5.2 Limitations of Offline Reinforcement Learning in World of Tanks Scenario

Offline reinforcement learning (RL) techniques, where an agent learns from a fixed dataset of previously collected experiences, offer several advantages such as sample efficiency and safety during training. However, when applied to complex and dynamic environments like the World of Tanks scenario, several limitations arise:

5.2.1 Limited Exploration

In World of Tanks, the environment is vast and complex, with a wide range of possible states and actions. Offline RL relies on the assumption that the collected dataset adequately covers the state-action space. However, due to the sheer size of the environment, it's challenging to collect a diverse dataset that captures all possible scenarios. This limitation can lead to poor generalization and suboptimal policies, especially in unseen or rare situations.

5.2.2 MDP formation

In this work, we have employed a Markov Decision Process (MDP) framework to model the World of Tanks, solely from the data available. . However, it is important to acknowledge that the MDP defined for our problem is incomplete and does not capture the full and accurate MDP of the problem.

5.2.3 Evaluation

As previously discussed in this thesis, evaluating the efficacy of offline RL policies is fraught with inherent challenges, largely stemming from the non-interactive nature of their assessment. The absence of real-time environmental interaction restricts our ability to directly observe how these policies would perform under actual conditions, presenting a significant barrier to comprehensive evaluation.

Complications Arising from Off-policy Evaluation (OPE)

The limitations of Off-policy Evaluation (OPE), a method explored in earlier chapters, become particularly salient in this context. While OPE allows us to estimate the effectiveness of a policy using data generated by a different policy, it is dependent on the assumption that the data sufficiently represents the scenarios that the new policy will encounter. This method, however, lacks definitive techniques and is still an active area of research, indicating the ongoing need for refinement to address its inherent biases and potential inaccuracies.

Limitations of Model-based Evaluation Techniques

Additionally, the dynamics model-based approaches discussed earlier are also subject to specific limitations. The reward function within these models, designed to simulate potential outcomes, does not accurately capture the complex and sparse reward structures characteristic of many real-world scenarios. These models, while useful, are indicative at best; they attempt to estimate performance based on simplified assumptions that may not hold true under varied or unexpected conditions. This discrepancy underscores the need for caution when relying on such models for policy evaluation.

As we conclude this thesis, it is clear that the evaluation of offline RL policies still faces considerable obstacles. Both OPE and model-based approaches require ongoing development to overcome their current limitations. Enhancing the fidelity of these evaluation techniques is crucial for advancing the field and ensuring that offline RL can be effectively applied in more

dynamic and unpredictable environments, such as World Of Tanks. The need for more sophisticated and accurate evaluation methods will undoubtedly be a focal point of future research in offline reinforcement learning.

5.3 Future Work

5.3.1 Enrich with more features

As discussed above, the information the agent sees as state, is limited and simplified, and does not capture the full state information needed, to be able to perform as well as possible. Incorporating richer state information, such as visual data, to provide the agent with a more comprehensive understanding of the environment. This would involve integrating sensory inputs that can help the agent perceive and react to its surroundings more effectively.

5.3.2 Enhancing Model Generalization

Improving the generalization capabilities of offline reinforcement learning models is crucial. Future efforts should aim at developing models that can adapt to the ever-evolving game dynamics, such as new tanks, maps, and other future updates.

5.3.2.1 Improving Dynamics Model for Evaluation

This is very important as the current dynamics model is an indicative model of the environment. Future work could include the creation of closer and accurate dynamics model (Transition Model, Reward Model) that more precisely captures the complexity and behaviour of the actual

environment, to get more proof results of the performance of the learnt policies. Or even better, to test these policies in an actual online environment.

5.3.2.2 Different Architectures

As this thesis has established a foundational approach to developing and implementing a robust dynamics model for offline reinforcement learning in strategic environments like World of Tanks, a promising direction for future work involves exploring more sophisticated neural network architectures such as the Q-Transformer [5, 39], which merges deep Q-learning with transformer-based models. The Q-Transformer architecture could leverage the transformer's capacity to understand complex sequences of actions and states due to its self-attention mechanisms, enhancing the model's ability to generalize from historical data and address challenges like distributional shift between training data and execution scenarios.

5.3.3 Multi-Agent Learning

Exploring collaborative multi-agent offline reinforcement learning, is a way to expand this work and introduce team-based strategies in World of Tanks. Future work could focus on using existing work on MARL, such as MADT [21], which implement a multi decision transformer model, or MA-CQL [8], which implements a multi-agent version of Conservative Q-Learning. MARL will enable policies that not only optimize individual player performance but also enhance team coordination and synergy, properties needed in a multiplayer game with team vs team scenarios and game modes.

5.3.4 LLM-powered Agents

With the recent development, of LLM-based agents ([37], but also inspiring from this work [31], where it states the potential benefit of using LLM pre-trained models in Offline RL scenarios, is natural to see how LLM-based agents can be build utilizing both Offline RL and already pre-trained LLMs models , to create a lifelong agent that can master different skills in World Of Tanks.

5.4 Conclusion

The application of offline reinforcement learning (RL) in a World of Tanks scenario presents several challenges and limitations. Despite these challenges and limitations, our analysis has shown that algorithms like CQL , IQL, DT are capable of producing significant returns under various discount factors, showcasing their robustness and adaptability in offline settings. This indicates that, while offline RL faces specific hurdles, there are promising avenues for achieving reliable and high-performance policy learning that least in some sub-tasks of the game, such as going to enemy base.

Moreover, offline RL exhibits properties such as improvement over the actions taken in the dataset, at least based on the estimated return on the dynamics model, and data stitching, where different behaviors are effectively combined to create more cohesive and optimized policies. These properties enhance the potential of offline RL to leverage diverse and suboptimal datasets to achieve superior performance.

Key limitations include the absence of actual online enviroment for evaluation the algorithms, , the presence of concept drift as the environment evolves, concerns regarding safety in competitive gameplay, and the limited MDP formation(lack of complete state information etc.).

Addressing these limitations requires innovative solutions that account for the unique characteristics of the World of Tanks scenario. Future research efforts should focus on developing techniques that enhance exploration, improve dataset quality, adapt to concept drift, use of more advanced architectures, multi-agent learning and use of more stable and robust evaluation methods.

Despite the challenges, offline RL remains a promising approach for learning in complex scenarios like World of Tanks. By overcoming these limitations, offline RL algorithms have the potential to enhance player experiences, optimize gameplay strategies, and advance the state-of-the-art in reinforcement learning in the gaming domain.

Bibliography

[1] World of Tanks – Legendary Online Multiplayer Tank Game.

[2] Arthur Argenson and Gabriel Dulac-Arnold. MODEL-BASED OFFLINE PLANNING. 2021.

[3] Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. TorchRL: A data-driven decision-making library for PyTorch, November 2023. arXiv:2306.00577 [cs].

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, June 2016. arXiv:1606.01540 [cs].

[5] Yevgen Chebotar, Quan Vuong, Alex Irpan, Karol Hausman, Fei Xia, Yao Lu, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, Keerthana Gopalakrishnan, Julian Ibarz, Ofir Nachum, Sumedh Sontakke, Grecia Salazar, Huong T Tran, Jodilyn Peralta, Clayton Tan, Deeksha Manjunath, Jaspiar Singht, Brianna Zitkovich, Tomas Jackson, Kanishka Rao, Chelsea Finn, and Sergey Levine. Q-Transformer: Scalable Offline Reinforcement Learning via Autoregressive Q-Functions.

[6] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling, June 2021. arXiv:2106.01345 [cs].

[7] Miroslav Dudík. Doubly robust policy evaluation and learning. *Proceedings of the 28th International Conference on Machine Learning*, pages 1097–1104, 2011.

[8] Eslam Eldeeb, Houssem Sifaou, Osvaldo Simeone, Mohammad Shehab, and Hirley Alves. Conservative and Risk-Aware Offline Multi-Agent Reinforcement Learning for Digital Twins, February 2024. arXiv:2402.08421 [cs].

[9] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for Deep Data-Driven Reinforcement Learning, February 2021. arXiv:2004.07219 [cs, stat].

[10] Justin Fu, Mohammad Norouzi, Ofir Nachum, George Tucker, Ziyu Wang, Alexander Novikov, Mengjiao Yang, Michael R. Zhang, Yutian Chen, Aviral Kumar, Cosmin Paduraru, Sergey Levine, and Tom Le Paine. Benchmarks for Deep Off-Policy Evaluation, March 2021. arXiv:2103.16596 [cs, stat].

[11] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array Programming with NumPy. *Nature*, 585(7825):357–362, September 2020. arXiv:2006.10256 [cs, stat].

[12] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[13] Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning*, pages 652–661, 2016.

[14] Athanasios Kapoutsis. The Monster of Distribution Shift in Offline RL and How to Pacify it, June 2022.

[15] Haruka Kiyohara, Ren Kishimoto, Kosuke Kawakami, Ken Kobayashi, Kazuhide Nakata, and Yuta Saito. SCOPE-RL: A Python Library for Offline Reinforcement Learning and Off-Policy Evaluation, March 2024. arXiv:2311.18206 [cs].

[16] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-Learning for Offline Reinforcement Learning, August 2020. arXiv:2006.04779 [cs, stat].

[17] Hoang M. Le, Cameron Voloshin, and Yisong Yue. Batch Policy Learning under Constraints, March 2019. arXiv:1903.08738 [cs, math, stat].

[18] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems, November 2020. arXiv:2005.01643 [cs, stat].

[19] S. Liu, K. C. See, K. Y. Ngiam, L. A. Celi, X. Sun, and M. Feng. Reinforcement learning for clinical decision support in critical care: Comprehensive review. *Journal of Medical Internet Research*, 22(7), Jul 2020.

[20] Fan-Ming Luo, Tian Xu, Xingchen Cao, and Yang Yu. REWARD-CONSISTENT DYNAMICS MODELS ARE STRONGLY GENERALIZABLE FOR OFFLINE REINFORCE- MENT LEARNING. 2024.

[21] Linghui Meng, Muning Wen, Yaodong Yang, Chenyang Le, Xiyun Li, Weinan Zhang, Ying Wen, Haifeng Zhang, Jun Wang, and Bo Xu. Offline Pre-trained Multi-Agent Decision Transformer: One Big Sequence Model Tackles All SMAC Tasks, June 2022. arXiv:2112.02845 [cs].

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013. arXiv:1312.5602 [cs].

[23] Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. Deep Dynamics Models for Learning Dexterous Manipulation, September 2019. arXiv:1909.11652 [cs].

[24] Alizée Pace, Hugo Yèche, Bernhard Schölkopf, Gunnar Rätsch, and Guy Tennenholtz. Delphic Offline Reinforcement Learning under Nonidentifiable Hidden Confounding, June 2023. arXiv:2306.01157 [cs].

[25] 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, July 2020. arXiv:2007.09055 [cs, stat].

[26] Doina Precup, Richard S Sutton, and Sridhar Mahadevan Dasgupta. Off-policy temporal-difference learning with function approximation. *Proceedings of the 18th International Conference on Machine Learning*, pages 417–424, 2001.

[27] Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–0, 2024. arXiv:2203.01387 [cs, stat].

[28] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. *European Conference on Machine Learning*, pages 317–328, 2005.

[29] Takuma Seno and Michita Imai. d3rlpy: An Offline Deep Reinforcement Learning Library, December 2022. arXiv:2111.03788 [cs].

[30] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A Survey of Deep Reinforcement Learning in Video Games, December 2019. arXiv:1912.10944 [cs].

[31] Ruizhe Shi, Yuyao Liu, Yanjie Ze, Simon S. Du, and Huazhe Xu. Unleashing the Power of Pre-trained Language Models for Offline Reinforcement Learning, October 2023. arXiv:2310.20587 [cs].

[32] B. Singh, R. Kumar, and V. Singh. Reinforcement learning in robotic applications: A comprehensive survey. *Artificial Intelligence Review*, 55:945–990, Feb 2022.

[33] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite, January 2018. arXiv:1801.00690 [cs].

[34] Philip S Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148, 2016.

[35] Masatoshi Uehara, Chengchun Shi, and Nathan Kallus. A Review of Off-Policy Evaluation in Reinforcement Learning, December 2022. arXiv:2212.06355 [cs, math, stat].

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. arXiv:1706.03762 [cs].

[37] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models, October 2023. arXiv:2305.16291 [cs].

[38] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[39] Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. Q-learning Decision Transformer: Leveraging Dynamic Programming for Conditional Sequence Modelling in Offline RL, May 2023. arXiv:2209.03993 [cs].

[40] Michael R. Zhang, Tom Le Paine, Ofir Nachum, Cosmin Paduraru, George Tucker, Ziyu Wang, and Mohammad Norouzi. Autoregressive Dynamics Models for Offline Policy Evaluation and Optimization, April 2021. arXiv:2104.13877 [cs, stat].

LIST OF TABLES

CQL Parameter Details	
Parameters	**Value**
Layers	2
Units per layer	256
Learning rate	3.0e-4
Optimizer	Adam
Hidden Layer Activation function	ReLU
Output Activation function(Critic)	Linear
Output Activation function(Actor)	Tanh
alpha (α)	controlled by optimizer
Lagrange-threshold (specific parameter)	10

Table 8: CQL Hyperparameter details

IQL Parameter Details	
Parameters	**Value**
Layers	2
Learning rate	3.0e-4
Units per layer	256
Optimizer	Adam
Hidden Layer Activation function	ReLU
Output Activation function(Critic)	Linear
Output Activation function(Actor)	Tanh
Expectile (specific parameter)	0.7

Table 9: IQL Hyperparameter details

DT Parameter Details	
Parameters	**Value**
Layers	3
Units per layer	512
Learning rate	1.0e-4
Optimizer	Adam
Num of Heads	1
Hidden Layer Activation function	ReLU
Output Activation function(Critic)	Linear
Output Activation function(Actor)	Tanh

Table 10: DT Hyperparameter details

BC Parameter Details	
Parameters	**Value**
Layers	2
Units per layer	256
Learning rate	0.001
Hidden Layer Activation function	ReLU
Output Activation function(Critic)	Linear
Output Activation function(Actor)	Tanh

Table 11: BC Hyperparameter details

DDPG Parameter Details	
Parameters	**Value**
Layers	2
Units per layer	256
Learning rate	3.0e-4
Hidden Layer Activation function	ReLU
Output Activation function(Critic)	Linear
Output Activation function(Actor)	Tanh

Table 12: DDPG Hyperparameter details