university of groningen

faculty of science and engineering

How Far Ahead Should One Look?

Offline Reinforcement Learning For Instructional Sequencing

Panagiotis Ritas

university of groningen / faculty of science and engineering

University of Groningen

How Far Ahead Should One Look?
Offline Reinforcement Learning For Instructional Sequencing

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at University of Groningen under the supervision of
Roelant Stegmann, MSc
Prof. Jelmer P. Borst (Artificial Intelligence, University of Groningen)
Prof. Matias Valdenegro Toro (Artificial Intelligence, University of Groningen)

Panagiotis Ritas (S3152529)

February 8, 2023

Contents

Acknowledgments

Firstly, I would like to thank Carlos, which with his fearless leadership capabilities managed to conceptualize this research project. Without Carlos, this research project would not exist in the first place. His impressive research skills managed to conceptualize from virtually zero knowledge into reinforcement learning into a concrete research project.
Secondly, I would like to thank Roelant for going out of his way to help me out at times when I needed it. Your guidance and compassionate leadership style have been invaluable for my journey toward this research project.
I would also like to thank Giorgos for his support. Our viewpoints and attitudes might clash in many ways, but we nonetheless (somehow?) manage to cooperate and support each other, and that has been tremendously important for me throughout this last year.
I would like to extend my thanks to my family. I'm forever grateful for your unconditional support, even if 2.2 thousand kilometers separate us.
I would also like to extend my thanks to my closest friends, living in Groningen or back in Greece. Your support or just even the simple act of spending time with you has helped me blow off steam, especially on the more challenging days.
I would also like to thank a certain group of ”goats” out there, which have all been nothing sort of compassionate and empathetic to each other as well as myself. My only regret is that I did not have the chance to see you physically as much, but on the times that I did its been nothing but nice.

Abstract

Deep reinforcement learning is a topic that holds great promise in solving a variety of optimization problems. Education and specifically the topic of instructional sequencing is one such problem, in which instructions, like feedback or exercises, are adaptively sequenced to a pupil in order to help them learn better. intelligent tutoring systems are computer-mediated systems that provide the infrastructure for educating and providing instructions to pupils.
Deep reinforcement learning has been extensively used for instructional sequencing, which aims to give the next instruction to the pupil in order to maximize learning over a horizon of instructions. It is not uncommon to also use models that are myopic, meaning that they aim to give the best next instruction to the pupil in order to maximize their learning. However, there has been little research on whether myopic or horizon-based models should be implemented in intelligent tutoring systems in order to induce instructional policies in pupils that maximize learning.
We aim to answer the question of finding the optimal horizon for a model look-ahead, and if myopic models are better than horizon-based models in order to maximize a pupil's learning. We trained two offline reinforcement learning (actor-critic) models with horizons of 10 and 20 on a dataset extracted from historical interactions of pupils from an intelligent tutoring system. The offline reinforcement learning models were evaluated on a designed simulator that aimed to simulate an intelligent tutoring system as well as its pupils. The original intelligent tutoring system upon which the simulator is based is used to educate pupils in primary schools in The Netherlands.
To answer the research question, the trained offline reinforcement learning models were ran in the simulator and compared against a myopic recurrent neural network with a horizon of 1 that historically served exercises in the intelligent tutoring system.
Results show that models with shorter horizons tend to induce better pedagogical policies in pupils, with the myopic recurrent neural network inducing the best pedagogical policies for the pupil's learning. The results also indicate that the first two served exercises to the pupil are crucial for the pupil's learning across a horizon of steps.

1 Introduction

In this section, we first introduce intelligent tutoring systems, instructional sequencing, Markov decision processes, and reinforcement learning. We explain and link these concepts so as to arrive at the research question we investigated for this thesis. Then, the research question and the significance of this study are then discussed. Finally, an overview of the research conducted for this thesis is given, as well as a brief outline of what each chapter of this thesis entails.

1.1 Disclosure

The effort of this thesis is part of a MSc graduation project conducted in a company, which will be kept anonymous for the remainder of this thesis. The company will be referred to as host company throughout the thesis.

1.2 Introduction

1.2.1 Intelligent Tutoring Systems and Theories of Instruction

In the last half-century, computers are increasingly prevalent in education. It has long been thought that computers can aid the education of pupils [1, 2]. For this, various Intelligent Tutoring Systems (ITS) have been developed: ITS are computer-mediated systems that provide instructional activities to a pupil in order to teach them a set of skills. ITS can ,for example, personalize a sequence of instructions or feedback for a pupil in order to help them learn better. The idea of infusing ITS with Artificial Intelligence (AI) exists since the 80s [3, 4, 5, 6]. AI can be used in ITS to learn the pupils' needs and personalize a sequence of instructions accordingly [3, 5, 7], which has been long known that it makes a difference on how well students learn [8].
ITS were first conceptualized in the early 1970s [6]. At the conception of ITS, researchers formed a new goal for computer-assisted instruction; ITS aim to replace the role of the human tutor in education. ITS aim to apply AI to realize the human tutor model into an "inteligent" computer-based tutor [6]. In this context, The goal of ITS would be to engage the students in reasoning activities and to interact with the students based on a deep understanding of a student's skills and behavior. Human tutors have been long known to significantly contribute towards a pupil's learning outcomes [9]. Achieving an "inteligent" tutor capable of doing the same tasks can drastically improve learning outcomes in pupils, and as a consequence, education.
As mentioned before, ITS aim to realize a human-tutor model into a computer-based instructor. For this to be achieved, a theory of instruction should be developed on the underlying ITS. Formally defined, theory of instruction is "a theory that offers explicit guidance on how to better help people learn and develop" [10]. A theory of instruction identifies what instructions should be like for teaching a pupil, and what are the possible choices of an educator in order to maximize student learning.
Richard Atkinson developed the first theory of instruction. Specifically, Atkinson formalized a set of rules for a theory of instruction towards a pupil, with the direct purpose of applying these rules when constructing an ITS [11]. This theory provides a framework for sequencing instructions in order to individualize learning in students. He claimed that a theory of instruction should include:

- Some model of how learning occurs (usually cognitive),
- A set of eligible instructions for the pupil,
- A set of objectives which the pupil wishes to learn,

- Some metric that defines the inherent value of an instruction assigned to the pupil.

Consider an ITS system with a model that serves exercises for a pupil when they are working on honing their mathematical skills. If we are creating such a system under the definition of Atkinson's theory of instruction, we need 1) a model that captures the pupil's mastery over mathematical skills, 2) a space of possible mathematical exercises that the ITS can give to a pupil, 3) A set of mathematical skills that the pupil wants to learn, like addition and subtraction, multiplication, division, etc, and 4) a positive reward given to the ITS. For example, a positive reward should be assigned to the Markov decision process when the pupil answers an exercise correctly.

1.2.2 Markov Decision Processes and Intelligent Tutoring Systems

Around the development of Atkinson's theory of instructions, the book ”dynamic Programming and Markov Decision Processes” was published by Ronald Howard [12]. This book proved to be one of the early foundational texts on Markov decision processes, with Ronald Howard now considered the ”father of decision analysis”. Markov decision processes provide the mathematical framework for modeling decision-making where there is a need to individualize instructions to a pupil. In fact, Ronald Howard's doctoral student, Richard Smallwood, was the first to apply the concept of Markov decision processes for sequencing the order of instructions in education [13]. We see then that Markov decision processes have been intertwined with modeling decision-making for educational purposes since their very conception. In fact, Atkinson's theory of instruction was originally formulated so that it can be mapped into Markov decision processes[11]; Since then, theories of instruction have been mapped into Markov decision processes many times, with the purpose of constructing ITS [14, 3, 7, 5].

1.2.3 Theories of Instruction and Tracing a Pupil's Knowledge

Atkinson's theory of instruction requires some model of how learning occurs in a pupil [11]. At the same time, one of the main goals of ITS is catering to a pupil's specific needs in order to help them to learn better [6]. An important component of ITS should therefore be to capture and adapt to a pupil's evolving knowledge under some subject. This will allow an ITS to keep track of a pupil's state of learning over time, and sequence instructions according to their current state of knowledge.
Researchers have been attempting to trace the evolution of a pupil's knowledge by applying dynamic models [15, 16, 17, 18]. This field is called knowledge tracing, and aims to create models that dynamically captures the evolving knowledge of a pupil over time [19]. In knowledge tracing models, a pupil's knowledge on a certain domain, say mathematics, is being captured through models that dynamically sequence the state of a pupil over time. Neural networks have been prevalent in the field in the last years, with function approximators like reccurent neural networks being used to dynamicaly capture the evolution of a pupil [15].
Recommender systems were traditionally used in platforms like Netflix, where users are recommended content like movies according to their preferences [20]. Under a recommender system in Netflix's platform, a user that watches comedies will be very likely recommended a comedic movie that he has not seen, instead of genres like drama. Recommender systems have also been applied to the field of knowledge tracing, especially matrix factorization methods, where a user's learning is dynamically captured under a specific subject, and exercises are embedded as vectors that describe certain qualities of the exercise, for example difficulty [21]. Matrix factorization methods then aim to combine a user's vector that captures its knowledge, and an exercise's vector to get a prediction of the user's probability to solve an exercise [16].

The vectors and predictions of a matrix factorization method can then be used to model how learning occurs in a pupil, as per Atkinson's theory of instruction. Subsequently, we can formulate the Markov decision process by using a matrix factorization method.

1.2.4 Reinforcement Learning For Sequencing Instructions

It was mentioned before that ITS are getting infused with AI. After all, the goal of ITS is to realise a computer ”tutor” that caters to a pupil's needs [1]. For this, reinforcement learning, one dominant branch of AI is extensively used in ITS [3, 22]. Reinforcement learning is the field of study in which an ”intelligent” agent is concerned with how to act in order to maximize the notion of cumulative rewards in an environment [23]. Reinforcement learning models are horizon-based because they take the next action in order to maximize the cumulative rewards over a sequence of actions. To give a concrete example, a horizon-based model would then aim to sequence the next, for example, 10 mathematical exercises to a pupil in such a way as to maximize the pupil's learning gains over the next 10 steps. In reinforcement learning models, this horizon can have a different lengths. Some might construct reinforcement learning models in an ITS that aim to maximize cumulative rewards over the next 10 steps, while others might choose a much longer horizon.
Reinforcement learning is one of the many options for sequencing instructions or exercises in an ITS. It is not uncommon to use myopic models for this [14, 24]. These are models that optimize the reward by considering the consequences of only the next action [5]. Such a model would then aim to serve a, for example, mathematical exercise such that it provides the biggest possible learning gains for a pupil over the next step. When considering how to construct a Markov decision process for sequencing the order of instructions in an ITS, using a myopic versus a horizon-based model is one of many design considerations. Regardless of the lookahead, this application of horizon and myopic models in education is called instructional sequencing, where the aim is to adaptively sequence instructions in order to induce pedagogical policies to help students learn better.

1.2.5 Mapping a Theory of Instruction to Markov Decision Processes

Research now on reinforcement learning and instructional sequencing is booming and different ITS are being developed that learn to construct pedagogical policies for pupils [3, 25]. Atkinson's theory of instruction has then been already mapped into Markov decision processes in many ways so as to construct these ITS [14, 3, 7]. Regardless, there seems to be little literature content on what are the best ways to realize Atkinson's theory of instruction in Markov decision processes. [13]. To give a concrete example: Atkinson's theory of instruction states that we need some model of how learning occurs in order to construct an ITS [11], but we also know that that it can be difficult to model a pupil's underlying learning and cognitive skills [26]. Many studies exist that model learning for pupil's in a certain way when constructing ITS. On the other hand, few studies exist that give evidence on which is the best way to model a pupil's learning when mapping this into Markov decision processes. In the next two paragraphs, we give two ways in which researchers have tried to mitigate this problem, by means of examples.
In a Markov decision process setting, A student might answer a question correctly. This does not mean that the pupil understands the question, but they might have answered correctly for the wrong reasons. Then, a student's answer is a noisy estimate of their underlying cognitive state or learning state. We can use a Partially Observable Markov Decision Process (POMDP) to account for this [27]. Under POMDPs, the agent cannot see the ”full” state of the pupil, but can only observe some approximation of it. It then learns to effectively give instructions to a pupil without having to know

their ”true” state. In other words, an agent that gives addition and subtraction exercises to pupil’s under a POMDP framework then does not know perfectly whether a pupil has mastered all skills of adding and subtracting; It gives exercises under the assumption that it approximately knows the state of the pupil. POMDPs then approximate reality better than Markov decision processes. This is especially relevant given that human tutors are bad judges of their pupil’s skills [28].

Other researchers mitigate the intractability of perfectly approximating a pupil’s cognitive skills or learning by constructing myopic models on the underlying Markov decision process [24]. In this case, The model suggests the best next instruction for the pupil. If the model suggests the best next mathematical exercise to serve to a pupil, it only has to approximate the pupils’ state of mathematical mastery over the next step and not over a horizon. It can be challenging to take into account how the pupil’s cognitive state might evolve over the next steps, and serving exercises based on a horizon might evolve compounding errors between the true and approximated cognitive pupil states. The more a model looks ahead, the more errors compound [29].

Above, we introduced two ways of mitigating the intractability of approximating a pupil’s true cognitive state. The point being illustrated is the following; many studies exist that model pupil learning into Markov decision processes in various ways, but no study exists which gives evidence for preferring the former over the latter.

In a similar fashion, the currently existing literature in instructional sequencing has been underwhelming in researching guidelines to follow when mapping theories of instruction into Markov decision processes. [5]. To the extent of our knowledge, there have been but a few authors that sought out what’s the best way to define a theory of instruction for the purpose of mapping it into Markov decision processes. For example, some suggest creating Markov decision processes with a small, restricted set of instructions to give to the pupil and to leverage psychological theories of learning into the models applied under the Markov decision processes, such as spaced repetition [30, 5].

1.2.6 Introduction to Research questions

In the scope of this thesis, we focus on the horizon that models constructed on top of Markov decision processes should consider. Specifically, we aim to study if myopic models improve learning outcomes for pupils, compared to horizon-based models. We aim to give future researchers guidance in constructing ITS, by understanding if horizon-based models work better than myopic models for instructional sequencing.

there has been a wide array of studies that implement horizon-based (reinforcement learning) models, as well as the contrast, myopic models [14, 3, 7, 5]. To the best of our knowledge, there are no studies that indicate whether researchers should be focusing on horizon or myopic approaches when applying AI models in an ITS for instructional sequencing settings.

1.3 Research Questions

The scope of this thesis is to answer the following research questions:

Q1. When inducing adaptive instructions towards a pupil, should horizon or myopic approaches be preferred for maximizing learning?

Q2. How far should a horizon-based approach look into the future for inducing instructional policies to the pupil?

1.4 Significance of the study

This study aims to provide evidence on the performance of myopic versus horizon-based models. This brings us closer to creating a more unified and concrete theory of instruction for pupils. To this end, future research can use the conclusions of this study to create more efficient and robust ITS. In a broader sense, this thesis aims to contribute to the betterment of technology-enhanced education.

1.5 Research overview

This thesis was developed on an ITS platform of the host company, tailored to primary school students. Adaptive learning serves as an ITS, and is an important feature of such platforms: Pupils are served exercises with no guidance from a teacher, under a subject, e.g mathematics, in order to maximize their learning. The exercises are usually served under a specific topic, e.g addition and subtraction. This thesis focused on offline reinforcement learning approaches, which can train purely on logged past interactions. To this end, we created a dataset for reinforcement learning training, which contained past interactions of various pupils with the platform. In this dataset, The pupils were served exercises under a myopic model, a recurrent neural network that we will call Behavioral Policy (BP) for the remainder of this thesis. The offline reinforcement learning models then aim to improve upon the BP by training on the dataset offline and with no further interaction with the environment.
Then, the dataset was used to train the reinforcement learning models offline. We created two reinforcement learning horizon variants: one with a 10-step horizon, and another with 20 step horizon. We call these models RL-10 and RL-20 respectively. Part of this training of these variants was defining the environment under a Markov decision process, and infusing the environment with knowledge from a matrix factorization model of the host company, called Knowledge Tracing (KT). Specifically, we defined the Markov decision process's state space, and action space, and shaped the reward under the KT model.
Afterward, a simulator was developed that mimics the adaptive learning platform of the host company. RL-10 and RL-20 were trained offline on the created dataset and then evaluated on the simulator. As the last step, we picked the best-performing reinforcement learning model for each horizon variant and compared them with the myopic recurrent neural network, BP. To be precise, we run the reinforcement learning variants against the simulator in order to gather the learning growth they induced in pupils. We compare this growth with the historically observed learning growth induced by the BP model. This allowed us to compare three models with different look-ahead horizons (1 vs 10 vs 20) and answer the two research questions posed in this thesis.
The steps taken to answer the research questions posed above can be summarised as thus:

- Extracted a dataset from the company's database for training (offline) reinforcement learning models.

- Used the dataset to train the two reinforcement learning horizon variants.

- Created a simulator that mimics the host company's ITS, or adaptive learning system. The simulator was used to 1) evaluate the trained reinforcement learning models, and 2) to evaluate (learning) growth induced to pupils by RL-10 and RL-20.

- Gathered the induced learning growth of the myopic BP that historically served exercises in the adaptive learning platform of the host company.

- Compared the induced learning growth of the myopic model with the learning growth of the different reinforcement learning horizon variants in order to answer the two research questions. To this end, the reinforcement learning horizon variants were run in the simulator and their induced learning growth to pupils was gathered.

1.6 Thesis Outline

In this chapter, we introduced the research topic of this study, the research questions, and the significance of this study. Chapter 2 Focuses on laying down the background for relevant topics. We first explain concepts like reinforcement learning, Markov decision processes, and offline reinforcement learning. Then, recommender systems, and specifically matrix factorization methods are laid down. Chapter 3 introduces the methodology: The underlying Markov decision process used in this study is formalized, and the reinforcement learning model used in all experiments is introduced, as well as the simulator. Chapter 4 introduces the two experiments performed in order to answer the research question, as well as relevant metrics. Chapter 5 presents the results of the two experiments. Finally, we conclude the thesis in chapter 6.

2 Background Literature

In this chapter we present the theoretical background that guided this study. In the first section, an overview of Reinforcement learning and Markov decision processes is given. In section 2.2, we then discuss deep reinforcement learning and the state-of-the-art in deep reinforcement learning algorithms. we then discuss offline (batch) reinforcement learning, and how it differs compared to online reinforcement learning. Finally, in section 2.4 we discuss recommender systems.

2.1 Reinforcement Learning

In this section, we first formalize Markov decision processes. Markov decision processes define the mathematical framework of the environment that an RL agent operates on. We then define reinforcement learning and formalize its goal. Following this, we discuss the fundamentals of different reinforcement learning algorithms: we discuss value functions, their significance in reinforcement learning algorithms, and the two different value functions used in reinforcement learning algorithms. We briefly mention model-free vs model-based reinforcement learning methods. We then discuss the bellman equation and its significance in solving Markov decision processes with reinforcement learning agents. Finally, on-policy and off-policy reinforcement learning methods are discussed. We discuss reinforcement learning so as to move on and understand deep reinforcement learning, as we used deep reinforcement learning algorithms in the scope of this thesis. We also discuss Markov decision processes as they are the backbone of decision modeling for reinforcement learning algorithms. We subsequently use Markov decision processes to define the mathematical framework of the environment under which the reinforcement learning agent will operate.

2.1.1 Markov Decision Processes and Reinforcement Learning Definitions

Reinforcement learning (RL) is the field of study in which an agent interacts with the environment and learns how it ought to act in order to maximize the notion of a cumulative reward [23]. In RL, an agent learns to take actions in complex environments by trial and error. While exploring the environment, the agent learns the ”goodness” of each action given the current state of the environment and learns to exploit already known actions. This allows the agent to learn the optimal action for each state of the environment, in order to maximize the notion of cumulative rewards over a horizon of steps.

Consider an RL agent that learns how to park a car: the agent will try different actions and observe their ”goodness” by observing e.g whether the car crashes somewhere. The goal is to learn to take the optimal action, e.g turn the steering wheel at the right angle, in each environment state, in order to learn how to park.

RL algorithms are typically formalized as Markov Decision Processes (MDP) [12]. MDPs define the mathematical framework of an environment under which an RL algorithm operates. Specifically, MDPs define the environment as a 5-tuple (S, A, P, R, γ) with state space $S = \{s_1, s_2, ..s_n\}$, action space A, $A = \{a_1, a_2, ..a_m\}$ scalar reward function $R = \{r_1, r_2, r_k\}$, transition dynamics function P , and γ is a discount factor, where $\gamma \in [0, 1]$ [23].

The state space typically defines all the different values an environment can take. The action space defines all possible actions an agent can take at a given time point, the reward function is used to associate the ”goodness” of an action under a specific metric, and the transition function P models how an environment might evolve when taking an action. γ is a constant used to control the importance of long-term rewards over rewards observed in a short-horizon [23].

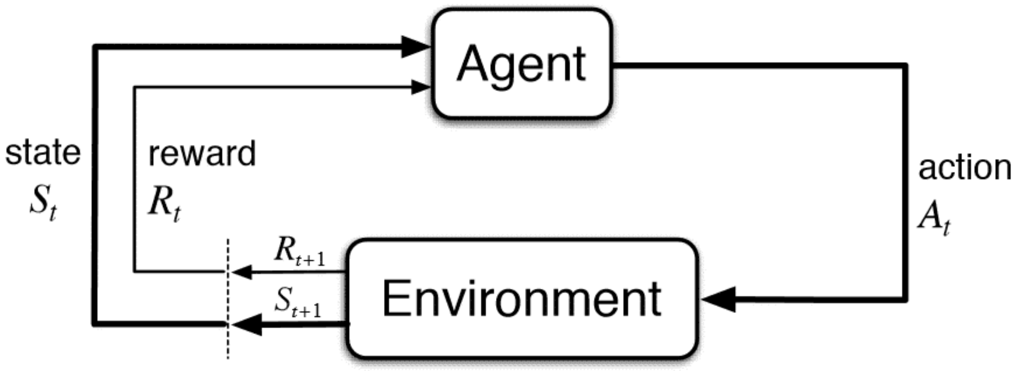

Figure 1: *Image outlining the interaction between the agent and the environment. The agent receives state s_t from the environment and acts upon it with action a_t. In return, it observes a new state s_{t+1} and the reward r_{t+1} associated with the taken action. Image credit: [23]*

The agent-environment interaction under an MDP is explained below: In the RL set-up, an autonomous agent for every time step $t = \{1,2,3...f\}$ observes a state of the environment s_t. The agent interacts with the environment and takes an action a_t where $a_t \in A$. The agent then observes a reward r_t and transitions to a new state of the environment s_{t+1}, usually a function of the current action and state s_t and a_t. The reward r_t is provided as direct feedback to the agent to indicate the quality of action a_t taken in state s_t. Transition dynamics function P defines the probabilities of the agent evolving from one state s_t to s_{t+1} [23]. The agent environment interaction is illustrated in Figure 1.
Over time, the agent learns the consequences of its actions. Exploration is a fundamental aspect of reinforcement learning, as the RL agent learns to explore new possibilities of undiscovered state-action pairs and to exploit known state-action pairs for learning an optimal policy.
Formally defined, the goal of an agent is to discover an optimal policy $\pi(a_t|s_t)$. An optimal policy aims to take an action that maximizes the returns over an episode in the environment. $\pi(a_t|s_t)$ yields the expected discounted return by:

$$G_t(s_t) = \mathop{\mathbb{E}}_{\pi}[\sum_{k=t}^{H} \gamma^t r_{t+1}] \tag{1}$$

$G_t(s_t)$ is defined as the episodic return, starting in a state s_t. H is the number of steps before an episode ends in the environment, and γ is the discount factor that ensures that returns stay finite and controls the importance between rewards in a short vs longer horizon. A higher γ distributes higher importance in long-term rewards and vice versa.
The agent's performance $J(\pi)$ is defined as the expected return $G_t(s_t)$ taken at a starting state over a horizon:

$$J(\pi) = \mathop{\mathbb{E}}_{s_0 \sim d_0}[G_0(s_0)] \tag{2}$$

Where d_0 is the starting distribution of states.

2.1.2 Value Functions

RL agents need a way to "quantify" the value, or "goodness" of the state that they are in, in order to take the best next action so as to maximize cumulative returns [23]. For example, consider an autonomous car; part of its action space should be how much to slow down or speed up at a given

time point. The underlying RL agent of the autonomous car should have a value function that assigns a low value to pressing the gas while being stuck in a traffic jam. In contrast, a high value should be assigned by the value function when the RL agent is first at a traffic light turning green.
Almost all RL algorithms involve estimating value functions: these value functions quantify either 1) the ”goodness” of an agent residing in a given state of the environment, or 2) the ”goodness” of an agent taking a particular action, given its the current state of the environment [23]. The notion of goodness is defined by what future cumulative rewards can be expected in a given state or state-action pair.
Value functions are important in RL as they are used to train an agent or to indicate to an agent which action to take, in order to maximize the notion of cumulative rewards, or better put, returns [23]. Value functions are estimated and corrected by the agent’s interaction with the environment.

Value-function definition We now move on with defining Q-functions. For every policy π there exists a value function that measures how good it is for a policy to be in a specific state, based on the expected returns. Two commonly used value functions for this in RL algorithms are the V and Q-function. The V-function $V^{\pi}(s)$ is defined as the estimated returns of a state s under π. The *Q-function*, denoted as $Q^{\pi}(s_t, a_t)$, defines the expected returns for taking action a_t, in a given state s_t, under a policy π which is followed further until the end of an episode:

$$Q^{\pi}(s,a) = \mathbb{E}[G(s_t)]|s_t = s, a_t = a] = \mathbb{E}[\sum_{k=0}^{H} \gamma^k r_{t+k+1}|s_t = s, a_t = a] \tag{3}$$

The optimal policy π^*, is obtained by maximizing the *Q-function* referred to above for all state-action pairs:

$$Q^*(s,a) = \max_{\pi} q^{\pi}(s,a) \tag{4}$$

This optimal policy can be realized by algorithms like policy iteration [31], with the underlying assumption that the agent has access to the reward and the transition functions R and P. In cases where R and P are not known, the underlying (S, A, R, P, γ) MDP can be solved with RL algorithms that implicitly learn the reward and transition function.
In such RL algorithms, the agent receives an initial state s_0 sampled from the distribution of the initial state d_0. The agent proposes an action a_t, according to the current state s_t. The environment returns the observed tuple (r_t, s_{t+1}, d_t), which denotes the reward signal at timestep t, where $t \in \{0, 1, ..n\}$, the following state at timestep t+1, and a terminal episode signal which indicates if the agent has reached the end of an episode. This agent-environment interaction provides the agent with information that it can use to realize the MDP [23].
Specifically, the agent can use the retrieved experience tuples $(s_t, a_t, r_t, s_{t+1}, d_t)$ from its interaction with the environment to implicitly learn the *Q-function* of the MDP. This introduces a family of RL algorithms called *model-free* methods, as they implicitly learn the *Q-function* and require no knowledge regarding the transition function P and the reward signal R of the environment [23]. In contrast, model-based RL algorithms exist which discard the notion of value functions and attempt to directly learn the distribution of the reward and transition functions R and P [23]. In this thesis we focus on a model-free method that learns the *Q-function* of the MDP.

Using the bellman equation to approximate the Q-function As mentioned in the previous paragraph, model-free methods exploit the retrieved experience tuples $(s_t, a_t, r_t, s_{t+1}, d_t)$ from the agent’s interaction with the environment. These experience tuples can be used to calculate the Bellman equation [32]. The Bellman equation uses a recursive property to create an estimate of the returns by calculating the Q-Function. Remember from equation 3 that the *Q-function* defines the expected returns by taking an action a_t, under state s_t. The Bellman equation is defined as:

$$\hat{Q}(s_t,a_t) = r_t + \gamma \hat{Q}^{\pi}(s_{t+1},\pi(s_{t+1})) \tag{5}$$

Where $\hat{Q}$ denotes the current estimation of the Q-function under policy π. The estimate of the returns is realized by bootstrapping the Q-function over the next observed state s_{t+1} from the environment. The Bellman equation and its recursive properties allow model-free RL algorithms to learn the Q-function of a policy, by defining it inside the Mean Squared Bellman Error:

$$MSBE = \frac{1}{N}\sum D||y_t - \hat{Q}(s_t,a_t)||^2 \tag{6}$$

where

$$y_t = r_t + d_t\gamma\hat{Q}(s_{t+1},\pi(s_{t+1})) \tag{7}$$

Recall the tuples of experiences $(s_t,a_t,r_t,s_{t+1},d_t)$, obtained from the agent's interaction with the environment as explained in paragraph 2.1.1. *MSBE* is learned by using all obtained tuples of experiences collected by the agent interacting with the environment. N is the number of experiences in dataset D, and in equation 7, $d_t = 0$ if at timestep t, an episode terminal was encountered. If an episode terminal is not encountered, then $d_t = 1$. *MSBE* is used to learn an estimator of the *Q-function*.

On-policy vs off-policy RL In this section, the difference between on-policy and off-policy RL algorithms is briefly discussed. The difference between on-policy and off-policy methods is illustrated in Figure 3. On-policy methods are RL algorithms that update the current estimation of the policy with data coming from the policy itself. One example of this is SARSA, in which the policy is updated directly by data coming from the policy itself [33]. In contrast, off-policy methods update a policy with data not necessarily coming from the same policy [23]. The data can come from a different behavioral policy and is contained in the form of e.g an experience replay buffer which will be discussed further later. In the scope of this thesis, we focus on off-policy methods. This is because our RL model trains offline with historical data from a different behavioral policy.

2.2 Deep Reinforcement Learning

In the sections below, we first discuss the shift from tabular to deep reinforcement learning methods in the last few years. We briefly explain the difference between tabular and deep RL. Then, recent state-of-the-art developments in the field of Deep RL will be discussed. First, three classes of Deep RL algorithms will be discussed: Value learning methods, policy learning methods, and actor-critic methods [23]. In value learning methods, the algorithm learns the value function of the MDP. In policy learning methods, or policy gradient methods, the algorithm learns and parametrizes the policy directly. The third class of RL algorithms are called actor-critic methods. Actor-critic methods are hybrid, meaning that they learn and parametrize both the value function and the policy. We first discuss value learning methods. Specifically, Deep-Q networks (DQN) are explained [34]. Then an overview of policy gradient and actor-critic methods will be given.
Following that, some state-of-the-art (SOTA) algorithms in deep RL under the actor-critic methods are formalized. We first discuss Deep Deterministic Policy Gradient (DDPG), leading up to the Twin Delayed Deep Deterministic gradient (TD3), which is a DDPG variant [35, 36]. This is done so as to understand how TD3 was developed, all the different underlying components that TD3 contains, and why it is the current state-of-the-art in the field.

2.2.1 Deep Reinforcement Learning Definition

RL, since its conception in the late 1950s, was formalized as a tabular method meaning that the value-function outputs are stored in a table[32]. In this case, the class of methods used to solve the underlying MDP is called dynamic programming. Using dynamic programming to solve a tabular method becomes infeasible when e.g the state space is very large or even continuous, in which states cannot be stored explicitly in a table.
In recent years, machine learning has brought renewed research in RL by introducing function approximators [34]. Using function approximators such as neural networks makes it possible to approximate the Q-function and the optimal policy of an MDP where the state space is continuous, as well as mitigate the intractability of storing large, or even infinite amount of values in a table. Although the neural networks are generally less ”deep” than in other domains, using networks to approximate the functions of the RL algorithm is now called Deep Reinforcement Learning (Deep RL).

2.2.2 State-of-the-art in deep Reinforcement Learning

Value Learning Value learning methods, a specific class of RL algorithms, aim to learn the value function of the policy [23]. In this way, the value function is parametrized (usually with a neural network), and the next action is taken according to this value function: we select the action that contains the highest value. One significant algorithm that falls under the scope of value learning RL methods is called Deep Q-network (DQN).

DQN Deep RL came to the forefront with a paper from Deepmind [34], in which the authors implemented function approximators to a Q-learning RL algorithm. The result was DQN, an off-policy, Deep RL algorithm that learned how to play Atari games.
DQN was the first RL algorithm that worked on a certain environment by implementing function approximators for learning the Q-function of the policy. Instead of storing the state-action pair values in a table the authors implemented a function approximator that learned to approximate the value function by sampling the historic experiences of the agent with the environment.
The function approximator parametrizes and optimizes the value function, in this case, the Q-function. Its output was an n-dimensional output space that estimated the returns for each available state-action pair. DQNs are off-policy methods, and adhere to the following greedy strategy:

$$a = max_a Q(s, a; \theta) \tag{8}$$

According to the equation above, DQNs navigate the environment by picking actions that have the maximum value-function, given the current state of the environment s, and the current function approximator's weights θ. At the same time, DQNs make sure to pick a different action occasionally so as to explore the state-action space and find the optimal actions.
Before, function approximators were difficult to implement in RL methods because of the problem of highly correlated experiences throughout the historic data. This problem was mitigated by the authors of DQN by using an Experience Replay (ER) Buffer. Gradient optimization techniques assume independent and identically distributed (i.i.d) estimates of the gradients. If this assumption does not hold, gradient optimization from e.g highly correlated samples that come from a single episode will lead to issues in training. The ER buffer is typically used in Deep RL methods to mitigate this issue. The buffer stores the historic experiences in memory and uniformly samples them during the training of the function approximators of the deep RL model, satisfying the assumption of i.i.d estimates of the gradient [37].

ER Buffers are used in most off-policy RL methods. While we discussed ER buffers for DQN, in the scope of this paper, all RL algorithms discussed have ER buffers implemented.

Policy Learning: Policy Gradient Algorithms As mentioned in 2.1.1, the objective of a RL agent is to maximize the "expected" reward when following a policy . There are various classes of RL algorithms that can be used for such a purpose, one of them being Policy Gradient (PG) methods. PG is a class of RL algorithms that aim to find the optimal behavior strategy by optimizing the policy directly. Intuitively, this means that PG methods directly learn the policy. This is done by parametrizing the policy. The policy is now a state-action function: given an input state, the policy outputs an action. while value learning algorithms like DQNs use the value-function to select an action, in contrast, PG methods parametrize the policy directly. This policy then directly maps a state to an optimal action [23].
The policy function $\pi_\theta(a|s)$ in PG methods outputs an action a, given a state s of the environment. It is parametrized with respect to θ [38]. PG methods use gradient ascent in regards to a defined local maximum, so as to optimize the policy parameters θ:

$$\Delta\theta = \alpha \nabla_\theta J(\theta) \tag{9}$$

Remember that DQN takes the max over the Q-values associated with all available state-action pairs in equation 8. In this case, the neural network of the Q-function the output layer represents each respective Q-value of all available state-action pairs. This approach then comes with difficulties when the action space is very large, as the function approximator ends up having a big output dimensionality for approximating the Q-values of all available state-action pairs. This approach even becomes intractable when the action space is continuous. PG methods in mitigate this problem by parametrizing the policy. They were introduced so as to allow RL algorithms to work with continuous action spaces [38].
In comparison to DQN, PG methods parametrize and learn the policy directly rather than learning the value function. Instead of choosing actions from the Q-value function approximator, PG methods implement a function approximator to the policy which learns to map a given state to the optimal action.
PG methods work well in high-dimensional, continuous action spaces and can be effective for learning stochastic policies. When the policy is parametrized and trained via gradient ascent in a stochastic environment, the policy implicitly learns the underlying randomness and noise from the MDP [35]. Instead of greedy action selection, the policy can now learn the associated probabilities of taking each action [38].

Policy and Value Learning: Actor-Critic methods Actor-critic methods were introduced so as to mitigate problems that occur in value and PG methods. Actor-critic methods mitigate a very particular limitation of PG methods: PG methods use equation 9 to update the policy weights at the end of an episode, using the returns gathered from this episode. By using the returns to update the policy, the algorithm misses valuable information from each step of a given episode, resulting in a high variance in the estimate of the gradient. In contrast, actor-critic methods mitigate this by learning during an episode: in this way, actor-critic methods obtain information on the value of each action and use that to update the policy. This results in a lower variance during the gradient update [39].
Actor-critic methods combine the ideas from value and policy learning to create a new hybrid approach. They parametrize both the value-function as well as the policy itself. The actor is the parametrized policy, which given a state, it outputs the action an agent should take. In contrast,

the critic approximates the value function for a state-action pair. Given a state-action pair, the critic outputs a value that approximates the returns for this action until the end of an episode. Both actor and critic are parametrized usually by function approximators like neural networks [23]. The critic and actor work in unison for policy optimization. Specifically, The critic evaluates the actions taken by the actor. In turn, the critic's approximations are used to optimize the actor's weights.
Below, we will be discussing the advancements in State-of-the-art of actor-critic methods. First, Deep Deterministic Policy Gradient (DDPG) will be discussed. Then, TD3 will be discussed, which is built on top of DDPG, with extra implementation tricks that allow it to obtain SOTA results in benchmark environments like Mujoco and Hopper [36] [40].

Actor-Critic Algorithm: DDPG DDPG is a model-free, off-policy RL algorithm. DDPG was created after the successes of DQN in playing Atari, as discussed in paragraph 2.2.2. DDPG aims to address some of DQNs limitations, namely that they work only on low-dimensional, discrete action spaces.
DDPG was initially created as a "DQN" for continuous action spaces. It uses a critic as a Q-function approximator $Q_\theta(s,a)$, and trains the critic using the MSBE in equation 6. Specifically, the MSBE is formulated as the expectation over the distribution by sampling over a dataset D of experiences. Vanilla DDPG's critic update is defined as follows:

$$L(\phi, D) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}[(y - Q_\phi(s,a))^2] \tag{10}$$

,where

$$y = r + d\gamma Q_\phi(s', \pi_\theta(s')) \tag{11}$$

Now, the experience tuple (s,a,r,s',d) in Equation 10 is sampled from the ER buffer D, making DDPG an off-policy method.
The ER buffer, parametrizing the critic, and updating it via the MSBE were all ideas inspired by DQNs [34]. In contrast to DQNs, the deterministic policy $\pi_\theta(s)$ is differentiable with respect to actions, which allows for gradient ascent with respect to the policy's parameters:

$$\underset{\theta}{max} \mathop{\mathbb{E}}_{s\sim D}[Q_\phi(s, \pi_\theta(s))] \tag{12}$$

The objective is to find the weights θ of the parametrized policy $\pi_\theta(s)$ such that the policy maps the state s to the action with the highest available Q-value.
In the next section, various adaptions off DDPG will be discussed that allow it to improve its performance as well as convergence properties during training time [35].

Target Networks and Polyak Averaging Recall the critic loss in equation 10. The Q-function is learned by bootstrapping the approximated Q-function in the next state. This can make training unstable, as this target becomes highly correlated with the optimized predictions because the two terms in the loss come from the same network. To mitigate the high correlation, the target usually comes from a separate target network, with parameters ϕ [34]. This allows the correlation between the MSBE terms to decrease and the target Q-network to stabilize. Consequently, it provides higher training stability. Training with target networks is achieved by modifying the MSBE loss in equation 13 as follows:

$$L(\phi, D) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}[(y - Q_\phi(s,a))^2] \tag{13}$$

,where

$$y = r + d\gamma Q_{\phi targ}(s', \pi_{\theta targ}(a')) \tag{14}$$

Where θ_{targ} and ϕ_{targ} are the target network weights. Target networks were first introduced in a DQN variant called DDQN [41]. They are an important implementation that helps DDPG converge toward the optimal policy.
Polyak averaging is also used when updating the target networks. Polyak averaging performs a linear interpolation when updating the target networks with the most recent copies of the actor and critic networks as follows:

$$\phi_{targ} \leftarrow \tau\phi_{targ} + (1-\tau)\phi, \tag{15}$$

$$\theta_{targ} \leftarrow \tau\theta_{targ} + (1-\tau)\theta, \tag{16}$$

where τ is the interpolation factor that affects the weight of delay for the copied target networks.

Exploration and Exploitation DDPG explores during training by injecting noise to its actions, typically from a zero-mean Gaussian distribution. The scale of noise is gradually reduced during training, as the agent explores less and exploits the state-action space more.
DDPG's training algorithm can be seen in the Apppendix in 6.5.

Actor-Critic Algorithm: TD3 We will now focus on Twin Delayed Deep Deterministic Policy Gradient (TD3): TD3 is an actor-critic, off-policy method that extends on DDPG with some tricks that enable better training convergence towards the optimal policy [36].
TD3 introduces several tricks to address limitations that occur in DDPG. The paper that introduced the model, called "Addressing Function Approximation Error in Actor-Critic Methods" concerns itself with a problem that occurs in DDPG: During training, the Q-function can start to dramatically overestimate Q-values of certain state-action pairs, and subsequently start exploiting the underlying Q-function errors. This can lead to a failure mode. This training failure is also called extrapolation error. This problem is especially prevalent in offline RL. We also touch upon this topic in paragraph 2.3.2.
To address extrapolation error, TD3 introduces three tricks built on top of DDPG 2.2.2: Clipped double-Q learning, policy smoothing, and "delayed" policy updates. These will be discussed in the following paragraphs.

Target Policy smoothing Recall from DDPG's critic loss that actions that are used to form the Q-target in the MSBE are based on a target policy $\pi_{\theta targ}$. In TD3, noise is also added to each dimension of the action. The action is then clipped to lie in the valid lower and upper action range. The target update is now defined as:

$$y = r + d\gamma Q_{\phi targ}(s', \pi_{\theta targ}(s') + \varepsilon) \tag{17}$$

, where

$$\varepsilon \sim clip(N(0,\sigma), -c, c) \tag{18}$$

an ε value has now been implemented in the target action that is typically sampled from a zero-mean Gaussian distribution. c is the lower and upper valid action range in which the new noisy action should lie in.

Injecting noise in the target has the benefit of regularizing the Q-function estimates across similar actions. That is, it enforces that similar actions should tend to have similar Q-values. Target policy smoothing prevents the algorithm from developing incorrect and sharp Q-value peaks, which are then exploited by the algorithm and consequently lead to extrapolating errors during the policy update.

Clipped double-Q learning Recall that DDPG uses a target network to calculate the target value in the MSBE critic loss. TD3 adds two Q-functions and two Q targets: Q_{ϕ_1}, Q_{ϕ_2}, and $Q_{\phi_{1,targ}}$, $Q_{\phi_{2,targ}}$ respectively. Only one update target is used by choosing the Q target network that provides the minimum value over the current gradient step:

$$y = r + d\gamma \min_{i=1,2} Q_{\phi_{i,targ}}(s', \pi_{\theta targ}(s')) \tag{19}$$

The Q-target that provides the minimum value is then chosen for the critic update.
The critic is now updated by regressing the minimum target over both Q-functions:

$$L(\phi_1, D) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}[(y - Q_{\phi_1}(s,a))^2] \tag{20}$$

$$L(\phi_2, D) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}[(y - Q_{\phi_2}(s,a))^2] \tag{21}$$

Using the smaller target for the Q-function update helps to fend off overestimation errors in the Q-function by choosing the less biased target network for the critic update. Choosing the smaller target might have the disadvantage of underestimation bias in the true Q-values, but that is much preferable to overestimating the true Q-values, as underestimation only leads to slower convergence to the optimal solution, rather than mode failures [36].

Delayed Policy Updates Compared to DDPG, TD3 updates the policy + target networks less frequently than the Q-function. By sufficiently delaying policy updates, the quality of the gradient ascent step increases when it occurs, as the value estimates consist of a lower variance. The policy and target network updates then happen at every indexed k gradient steps of the critic update. Similarly to DDPG, target network updates happen by means of Polyak averaging as explained in paragraph 2.2.2.
TD3's policy updates nevertheless are almost identical to DDPG:

$$\max_{\theta} \mathop{\mathbb{E}}_{s\sim D}[Q_{\phi 1}(s, \pi_\theta(s)] \tag{22}$$

Where in TD3 only the Q-function with weights ϕ_1 is maximized [36].

Exploration and Exploitation Exploration in TD3 is identical to DDPG, where noise from an uncorrelated zero-mean Gaussian distribution is sampled to its actions. Exploration is out of the scope of this thesis, the implementation of TD3 was done in an offline setting with a static dataset.
The pseudocode outlines TD3's training procedure.

Algorithm 1 Twin Delayer Deep Deterministic Policy Gradient (TD3)

```
1: Input: initial policy parameters $\theta$, Q-function $\phi_1, \phi_2$,replay buffed D with logged data from behavioural policy $\pi_\beta$
2: set target parameters equal to main parameters $\theta_{targ} \to \theta$, $\phi_{targ,1} \to \phi_1$, , $\phi_{targ,2} \to \phi_2$.
3: **repeat**
4:     Observe state s and select action $a = clip(\mu_{\theta(s)} + \varepsilon, a_{Low}, a_{High})$, where $\varepsilon \sim N$
5:     Execute a in environment
6:     Observe next state s', reward r, and done signal d to indicate whether s' is terminal
7:     Store (s,a,r,s',d) in replay buffer D
8:     if s' is terminal, reset environment state
9:     **if** its time to update **then**
10:        **for** j in range(however many updates) **do**
11:            Randomly sample a batch of transitions, $B = \{(s,a,r,s',d\}$ from D
12:            Compute target actions
                   $$a'(s') = clip(\mu_{\theta targ}(s') + clip(\varepsilon, -c, c, a_{Low}, a_{High}), \varepsilon \sim N(0,\sigma)$$
13:            Compute targets
                   $$y = r + \gamma(1-d) \min_{i=1,2} Q_{\phi targ,i}(s', a'(s'))$$
14:            update Q-functions by one step of gradient ascent:
                   $$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} (Q_{\phi i}(s,a) - y)^2, i = 1,2$$
15:            **if** j mod policy delay = 0 **then**
16:                Update policy with gradient ascent:
                       $$\nabla_\theta \frac{1}{|B|} Q_{\phi 1}(s, \mu_\theta(s))$$
17:                Update target networks with
                       $$\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1-\rho)\phi, i = 1,2$$
                       $$\theta_{targ} \leftarrow \rho\theta_{targ} + (1-\rho)\theta,$$
18:            **end if**
19:        **end for**
20:    **end if**
21: **until** convergence
```

2.3 Offline Reinforcement Learning

In this chapter, offline RL will be explained in detail. We will then address the problem of distributional shift, a prevalent issue that makes offline RL difficult to apply. After, solutions to distributional shift will be addressed. Lastly, imitation learning and specifically behavioral cloning will be discussed, which is a form of supervised learning that can be used as a solution to the problem of distributional shift.

Offline RL is important in the scope of this thesis as our model is trained on offline data, logged from the pupil's past interaction with the environment. In this case, no exploration is done, and the algorithm aims to improve upon the observed behavior of the historical data. The offline RL algorithm used in this thesis is a TD3 variant, called TD3-BC, that uses behavioural cloning to mitigate distributional shift [42].

2.3.1 Offline Reinforcement Learning Definition

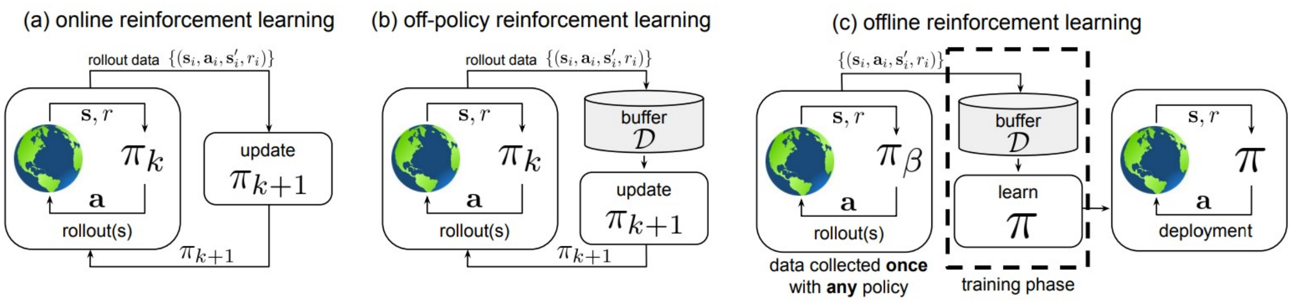
(a) online reinforcement learning
rollout data $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$
$\mathbf{s}, r$
π_k
$\mathbf{a}$
rollout(s)
update
π_{k+1}
π_{k+1}
(b) off-policy reinforcement learning
rollout data $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$
buffer
$\mathcal{D}$
update
π_{k+1}
$\mathbf{s}, r$
π_k
$\mathbf{a}$
rollout(s)
π_{k+1}
(c) offline reinforcement learning
$\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$
$\mathbf{s}, r$
π_β
$\mathbf{a}$
rollout(s)
data collected **once** with **any** policy
buffer
$\mathcal{D}$
learn
π
training phase
$\mathbf{s}, r$
π
$\mathbf{a}$
deployment

Figure 2: Difference between the on-policy (Online), off-policy, and offline RL paradigms: a) in online RL, the policy π_i is updated using the data collected by the policy π_i. b) in off-policy RL, an agent's experiences are appended into an experience replay Buffer D. D consists of a number of constantly updated policies $\pi_0, \pi_1 ... \pi_i$ exploited to find a new policy π_{k+1}. c) in Offline RL, the experience replay buffer D is static and is usually collected by some unknown behavioral policy. Offline RL training aims improve upon the behavioral policy without explicit interaction with the real environment. Image credit: [43].

Offline RL is a topic that gained traction in recent years along with the age of big data, where massive quantities of data are available to be exploited [44]. It thrives in domains where data collection can be expensive or where online exploration is dangerous, and where an under-performing policy can be a cause for considerable harm (e.g robotics, healthcare). Previously, this problem was mitigated by designing high-fidelity simulations on which an RL algorithm can train. This approach has the obvious issues that designing a very accurate simulator can be costly and that the trained policy will still be sub optimal when deployed in the real world [45]. Offline RL is also named data-driven RL, where historical data are exploited in order to obtain a policy that improves upon a behavioral policy π_β without any explicit interaction with the environment.
Offline (Batch) RL breaks the assumption that the agent can and should interact with an environment to learn an optimal, or at least better policy. In Batch RL, the dataset on which the agent is trained has been instead collected by one or multiple historical behavioral policies [43]. The goal of the agent is to learn an optimal policy π^* using a static dataset, defined as the tuple $D = \{(\overset{i}{s_t}, \overset{i}{a_t}, \overset{i}{r_t}, \overset{i}{s_{t+1}})\}$. The objective of Batch RL is therefore to maximize Equation 1 while having no further interaction with the environment. In essence, the objective of batch RL is inferring an understanding of the dynamical processes that underlie an MDP, while not interacting with the dynamical system in an MDP formulated way, but only a fixed dataset. Using this fixed dataset, an optimal policy π^* is derived from a past logged behavioral policy π_β that solves the underlying MDP and obtains the maximum possible returns during test time [43]. Offline RL can be thought of as a family of algorithms that are off-policy variants: in off-policy scenarios, the agent's experiences are appended to a data buffer, usually an ER buffer, in which various policies are observed and are exploited to learn an updated new policy. In contrast, in offline RL a dataset D is collected from some possibly unknown behavioral

policy, and is not altered during training. Figure 3 outlines the differences between online, off-policy, and offline RL.

2.3.2 Challenges in Offline RL

Distributional Shift The main component that makes offline RL difficult is that it makes queries that lie out of the scope of the defined dataset. Consider a standard supervised learning problem: The goal is to train a function approximator that attains a good performance in the underlying distribution that the training data comes from. Offline RL agents on the other hand make "what if" questions on what will happen if the agent takes an action that is different from what is observed in the data, in order to outperform the behavioral policy π_β This problem is called distributional shift, in which the offline RL agent, due to its changes in the visited states, is evaluated on a different distribution observed in the static dataset.
Consider the actor-critic methods of model-free RL algorithms discussed in 2.2.2. These algorithms implicitly learn the Q-function by exploiting the bellman equation 5. Under the scope of the actor-critic methods are DDPG and TD3, discussed in 2.2.2, and 2.2.2 respectively. We will be explaining the problem of distributional shift only in terms of TD3, as this is the model used in this thesis.
In the offline setting, divergence occurs between the true value of the value function and the estimated value function by the algorithm. Divergence specifically occurs because of bootstrapping and the approximator 5: when some Q-function estimates are incorrectly high, the Q-function approximator overestimates the returns of some actions, which is picked up during policy updates. For TD3, this accumulation error happens in equation 20. The overestimation error occurring in the Q-values is then propagated on the policy maximization step as shown in equation 22. In the offline setting, this causes the policy to visit states for which there is no information on the dataset, and creates a vicious circle that causes training to fail. This is usually mitigated to some degree in online RL algorithms: exploration allows the agent to visit new states and sample more information on them, eventually correcting the overestimated Q-values [36].
Methods in the literature that tackle the problem of distributional shift focus on constraining the policy from diverging from the behavioral policy π_β. These can be formalized as explicit and implicit policy constraint methods. We briefly discuss these methods below:

Implicit policy constraint methods In implicit policy constraint methods the distributional shift problem is addressed by modifying the policy improvement objective to a constrained objective we wish to obtain:

$$\underset{\phi}{argmax} \underset{(s,a)\sim D}{\mathbb{E}}[log(\pi_\phi(a|s))f(Q_\theta,\pi_\phi,s,a)] \tag{23}$$

In this form of policy constraint, the policy is constrained in a weighted manner: f is a filtering function that evaluates state-action pairs in the replay buffer. Function f pushes the policy towards experiences in the replay buffer that promise good rewards, while bad actions are filtered and discarded.
An example algorithm that is included under implicit policy constraint methods is AWAC [46].

Explicit policy constraint methods In explicit policy constraint methods, the parametrized policy π_θ is forced to stay close to the behavioral policy π_β under some explicit divergence measure D_m.

$$\underset{\phi}{argmax} \underset{s\sim D}{\mathbb{E}}[\underset{a\sim\pi_{\phi}(.|a)}{\mathbb{E}}[Q_{\theta}(s,a)]]s.t.D_m(\pi_{\phi},\pi_{\beta}) \leq \varepsilon \tag{24}$$

In this family of methods, The policy is constrained to act close to π_{β}, making these methods a variant of imitation learning. In that sense, the behavioral policy π_{β} is considered an expert demonstrator. Behavioral cloning divergence measures can be implemented under these methods.
This family of algorithms is ideal for the scope of this thesis, as the behavioral policy of the dataset is an expert demonstrator. It is the best-performing model under the machine learning methods employed by the company in the platform. We therefore, used an offline RL algorithm that mitigates distributional shift by explicit policy constraint. Specifically, TD3-BC uses a behavioral cloning regularization term during policy updates to stay close to the behavioral distribution of the behavioral policy π_beta. More in imitation learning and behavioral cloning is explained in 2.3.3. TD3-BC is explained in paragraph 3.3.

Offline RL evaluation Another significant challenge that is emphasized in the offline RL literature is the question of how an agent's performance should be evaluated. In standard RL, The policy is usually tested in the real world. This is potentially dangerous, as it can be dangerous to deploy a policy that might cause considerable harm if it underperforms. Deploying a sub optimal policy for instructional sequencing in pupils is dangerous, as the pupils will be exposed to sub optimal teaching, and will therefore fail to learn efficiently.
To this end, several methods of evaluating policies purely offline have been developed in recent years, which are called Off-Policy Evaluation (OPE) methods. For more on the topic, see [47]. OPE methods are not discussed further as part of this thesis entails performing policy evaluation on a simulator.

2.3.3 Imitation Learning

Imitation learning can be considered similar to RL. However the two approaches differ in that imitation learning algorithms do not leverage a reward function, but instead directly learn the policy from a set of demonstrations [48]. The set of demonstrations is generated by an expert policy and the imitation learning algorithms directly aim to imitate the policy. The approaches that aim to directly imitate the policy are called Behavioral Cloning (BC) algorithms.

Behavioral Cloning BC methods use a set of trajectories $\tau \in T$ to define a policy π which directly imitates the behavior observed in τ. To do this, BC is formulated as a supervised learning problem where the goal is to minimize the difference between an optimized policy and the expert policy:

$$\hat{\pi}^* = \underset{\pi}{argmin} \sum_{\tau\in T} \sum_{x\in T} L(\pi(x), \pi^*(x)) \tag{25}$$

Where L is the loss function, $\pi^*(x)$ Is the expert demonstrator's action at a certain state x, and $\hat{\pi}^*$ is the policy we aim to find. It can be observed that behavioral cloning directly imitates the expert π behavior, whereas RL algorithms explore the state-action space combinations to find an optimal policy. The combination of the two approaches in the offline RL setting is thought to be promising for mitigating distributional shift. Specifically, BC has been used as a regularization term for policy optimization in DDPG [35] as well as other PG methods [49]. BC combined with offline RL can encourage the RL algorithm to take actions that stay close to the actions observed in the dataset, therefore minimizing distributional shift.

In the scope of this thesis, experiments conducted use an algorithm called TD3-BC, which modifies the TD3 algorithm mentioned in 2.2.2 with a BC regularization term. TD3-BC and its components are explained in the methods section.

2.4 Recommender Systems

In this section, we will be giving a general overview of recommender systems. We first define what a recommender system is. Then, we define collaborative filtering, a recommender system method for finding suggestions for users. We then explain neighborhood methods and latent factor models, two methods under collaborative filtering. After that, matrix factorization models are discussed, which are a certain type of latent factor model. We lead up to matrix factorization, and especially matrix factorization and knowledge tracing, so as to describe how KT works.
A trained matrix factorization method for Knowledge Tracing (KT) will be used to describe the state and action space of the underlying MDP in our problem specification. KT will be also used to shape the reward of the MDP.

2.4.1 Recommender Systems Overview

A recommender system consists of an algorithm that suggests items to users, where the items should be of interest to the users [50]. Specifically, a recommender system falls under the class of information filtering systems, of which the aim is to give recommendations to a user, for a number of items that are relevant for them [50]. Recommender systems are used in engines such as Netflix, in which users are suggested movies that they might like based on past preferences. The items can range from a number of different categories, from movies to songs, and in the case of this thesis, exercises to serve a pupil. A recommender system aims to simplify item selection for a user by recommending a series of items that the user might have the most interest in. In this context, a Netflix user that has watched drama movies in the past will be very likely recommended drama movies they have not seen before.
Recommender systems can be designed in various ways. As mentioned in the definition, a recommender system aims to find item suggestions for a certain user. One way to do this for a user is by finding other users that rated items similarly as the user of interest. Then, the recommender system suggests an item to the user that they have not engaged with but the other similar users did. This approach of recommender systems is called collaborative filtering [51].
Collaborative filtering methods can be divided into two parts: *neighborhood methods* and *latent factor models*. Neighborhood methods compute the relationship between items. In this way, the user's preference for an item is based on ratings of nearby items by the same user. The neighboring items are embedded in a way that they get similar ratings when rated by the same user [20]. To give an example: consider a product suggestion from an online retail shop. A vacuum cleaner should have neighbours that are similar, so for example home appliances or brooms. To predict a user's rating for the vacuum cleaner, we can look at the nearest neighbours that the user rated.
In contrast, latent factor models aim to explain the rating of a user by defining items and users on factors inferred from the rating patterns [20]. Consider the example illustrated on Figure 4, for an online retail shop that rates its items into two factors, entertainment and efficiency. A vacuum cleaner might be efficient, but not as entertaining, and it would therefore score higher on efficiency. A video game console will likely score high on the entertainment factor but not as much on efficiency. Users are also defined along the same factors. For the user, each factor measures how much a user ”prefers” the corresponding factor. A user that scores high on the efficiency factor and lower on the entertainment factor would therefore be more likely to be suggested a vacuum cleaner compared to a video game

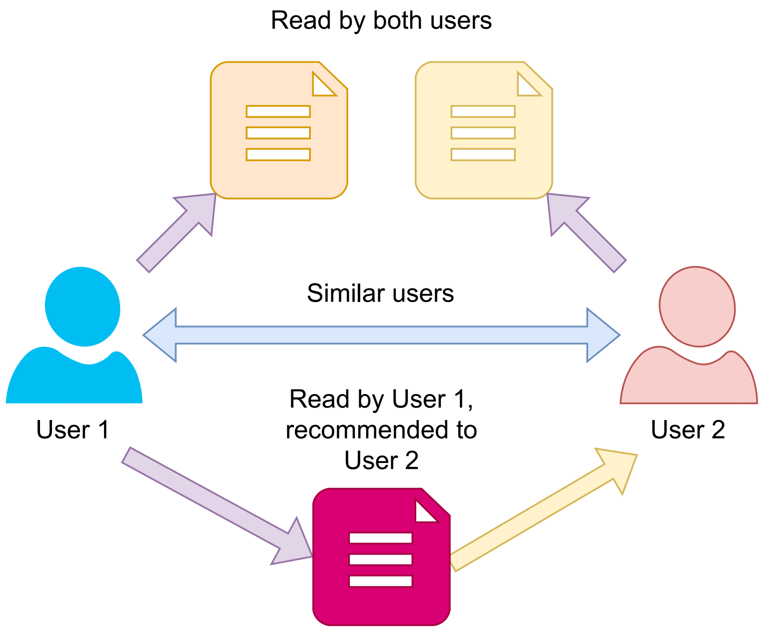

Figure 3: *Figure outlining collaborative filtering methods. In this approach, a user 2 is recommended content that similar user 1 has engaged with. The intuition of this approach is that if users are similar, then their preferences should also be similar.*

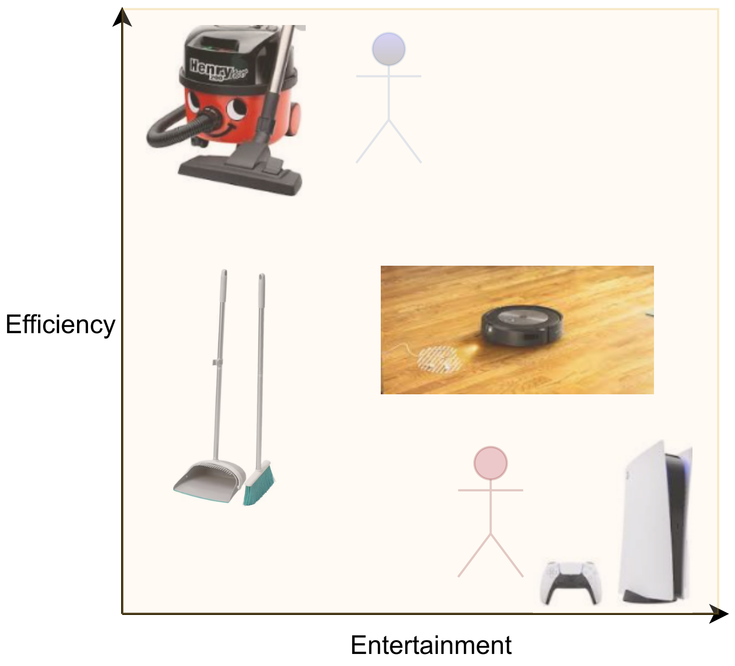

Figure 4: *Depiction of the latent factor model. In this model, two well-defined factors are plotted along the axes. Both items (broom, video game console), as well as users, are defined along the factors. In the latent factor model, a user that e.g rates high on entertainment but low on efficiency is more likely to be suggested a video game console compared to a broom.*

console. Note that these factors could be well defined, as illustrated in the example above, but the defined factors can also be uninterpretable [51].

2.4.2 Matrix Factorization Models

Matrix Factorization (MF) models are methods categorized under the latent factor models. They are considered to be one of the most successful applications of latent factor models [20]. MF methods also characterize both items and users by vectors of factors inferred from item rating patterns.
MF methods map users and items to a latent factor space. Interactions between user and items happen as inner products of that space. For each item and user, there exists a vector $e \in R^k$, $e \in E$ and $c \in R^k$, $c \in C$. The vector e for an item characterizes the proportion of which this item contains the characteristics of each latent factor. For a user, the vector c characterizes the amount in which a user is interested in each latent factor.
These vectors are then set up in such a way that a user's interest in a certain item is extracted by taking dot product between a user and item vector. The dot product d_{ui} between a user u and an item i is then defined by

$$d_{ui} = e_i^T c_u + b \tag{26}$$

where b is a bias $\in R^1$ that accounts for variations in rating values. When a reccomender system has mapped vectors $e \in R^K$ and $c \in R^k$, then a user's preference for an item can be obtained by using equation 26.

Matrix factorization for knowledge tracing Knowledge tracing is the task of modeling student knowledge over time so that we can accurately predict how students will perform in future interactions [15]. MF methods have been used for knowledge tracing [16, 21]. Previously, we were discussing MF methods with examples under product recommendations to users. A user's latent factors are now defined as skills to be mastered by the user. These factors can be defined as specific skills (e.g mastery over addition and subtraction, multiplication, etc), or could be uninterpretable. At the same time, items are now replaced by exercises, and their latent factors describe some characteristic of the exercise; e.g, the exercise is embedded in a 10-dimensional space, where each factor describes some aspect of the difficulty of the exercise.
For each item and exercise, there exists a vector $e \in R^k$, $e \in E$ and $c \in R^k$, $c \in C$. The latent factors for the user's vector e now describe the knowledge of a pupil under a certain subject (e.g, mathematics). A c vector for a given exercise describes the difficulty of an exercise by embedding it. In this way, exercises with similar difficulty are closely embedded in an n-dimensional space.
Given these two vectors, we can now model the students' knowledge over time. We can use Equation 26 to predict a pupil's answer on a given exercise, assuming vectors e and c.

3 Methods

In this chapter, we discuss the different components developed in this thesis, in order to run experiments.
First, we discuss the underlying RL problem formulation. In this section, we formalize the underlying MDPs state S, action space A, and reward function R. The MDP is formalized on top of the KT model, and the RL algorithm is built upon this MDP and tries to solve it. Details on how the MDP space was formalized are elaborated upon in section 3.2.3. The dataset and the behavioral policy behind the dataset are also discussed. The behavioral policy comes from a recurrent neural network deployed in the company's platform, which served exercises to pupils, namely BP. BP was first discussed in chapter 1.5.
As a next step, we discuss the RL model TD3-BC which is used in this thesis. TD3-BC is a variant of TD3, made to work in offline settings [42]. We specifically discuss the extra implementation on top of TD3 that makes TD3-BC, namely the behavioral cloning regularization term. Last, the simulator that was developed as part of the thesis will be explained, which aims to imitate the company's learning platform, and is used to run experiments on with the RL models. We discuss the simulator environment, as well as provide pseudocode on how it works. We also discuss simulating the pupil's response in the simulator and discretizing continuous action spaces into discrete exercises with the Wolpertinger architecture [52].

3.1 Reinforcement Learning Problem Formulation

We aim to train an RL model to learn an optimal policy π, defined as a mapping from state space S to action space A. The policy should improve upon the logged behavioral policy π_β, or BP, defined in the dataset. Given any state, the RL agent follows a series of actions that will maximize the expected returns over a series of steps. In order to train offline and validate the policy's performance in the simulator, the RL problem needs to be defined under an underlying MDP. To this end, The MDP is formalized as an experience tuple (s,a,r,s',d).

State Space: $S \in \mathbb{R}^{154}$. The state space is a continuous vector describing a pupil's knowledge under a specific subject, in this case, mathematics. All states $s \in S$ are strictly positive with no defined upper range. Their features usually take values of up to 0.7, and each feature is uninterpretable. The pupil state is calculated under KT. More on KT and how the pupil states were obtained is explained in 3.2.3.

Action Space: The action space is a closed set of continuous vectors. Each vector reflects an exercise under the topic of addition and subtraction in the subject of mathematics, where $A = [a_1, a_2..a_n] \in \mathbb{R}^{154}$. All $a_i \in A$ are discrete exercises embedded into 154-dimensional vectors according to their difficulty, and each feature embedding is strictly positive with no defined upper range. Each feature of an exercise embedding a_i is uninterpretable. Their features usually take values of up to 0.7, but the highest-defined feature observed in a given vector $a \in A$ has a value of 1.7. While the action space is "discrete" in the sense that it consists of a closed set of continuous vectors, the RL agent learns to output continuous actions called proto-actions, or $\hat{a}$, where most likely $\hat{a} \notin A$. The RL agent maps these proto-actions into action space A and transforms them into a discrete exercise by choosing the "best" neighboring vector $a_i \in A$. More on how the RL agent learns to take actions in the simulator is explained in the Wolpertinger architecture section 3.4.2.

Reward Function: The reward function is defined as the Delta Average Solve Probability (DASP) for a specific skill in the topic of addition and subtraction. Intuitively, this reward function aims to capture the ”leap” in the growth of a pupil under a specific skill that the pupil aims to learn. For example, that skill could be learning to count up to 20. More on DASP can be found in 3.2.3.

Finally, s’ is the next observed pupil state under the KT model, given by Equation 27. d is the episode terminal signal which indicates whether the student has reached the end of the episode.
Given how the RL problem is formulated, each specific component of the MDP will be laid down in detail in the following sections.

3.2 Dataset

The dataset that was gathered in this thesis will be explained in the section below. The dataset was used to train all TD3-BC models offline. The TD3-BC model is explained further in 3.3.
In this section, we first briefly discuss the Behavioral policy BP that historically served exercises to pupils. We then discuss the dataset in the context of the host company. Then, we explain in detail how we formalized the underlying MDP existing in the dataset, as well as the KT model used for this. Last, we discuss in section 3.2.4 how we manipulated the dataset in order to create RL variants that look ahead in different horizons.

3.2.1 Logged Behavioural Policy

The dataset was acquired from logged pupil traces of a deep knowledge tracing model called BP, used in the platform to serve exercises to pupils. BP is a recurrent neural network that aims to capture the skills of a pupil under a specific skill in the topic of addition and subtraction. It serves exercises to pupils by calculating a solve probability for each exercise under the topic in which the pupil learns. The exercise which is the most impactful with a solve probability around 0.8 is chosen. This is done so as to serve an exercise to the pupil that is challenging enough for them to learn, but also not too hard at the same time.
BP is myopic and chooses the best next action in order to maximize growth in pupils. The RL variants will be fit on experience tuples of (s,a,r,s’,d), where the exercises were historically served by the BP, in order to create RL variants that look ahead in a horizon. The RL models then aim to train on past experiences from the BP in order to acquire a policy that improves upon the behavior of the BP.

3.2.2 Dataset formalization

Here we explain the structure of the dataset, and how it relates to the host company’s platform.
The dataset used to train the RL models in this thesis contains past pupils’ interactions with the company’s learning platform. The platform consists of the following hierarchy: **Subject**, which contains a different number of classes, e.g mathematics or spelling. Under subjects are different subcategories called **topics**, which contain specific lessons under the given subject. For example, one topic could be addition and subtraction of numbers. Finally, under topics, **skills** are defined, in which a specific objective is defined to learn under a topic. An example of a specific skill is the skill of ”addition and subtraction up to twenty by heart”, which is contained under the topic of addition and subtraction.
The dataset contains logged interactions only under the topic of addition and subtraction. Addition and subtraction contain 81 different skills, for example, ”addition and subtraction up to twenty by

heart", another skill to be learned could be "addition and subtraction up to 50". Addition and subtraction is defined under the subject of mathematics. This topic was chosen as it is the largest topic under the company's learning platform in terms of pupil engagement.
The logged dataset included answers from pupils from two functionalities of the company's learning platform: 1) lessons, in which the pupils answer an exercise under the guidance of the teacher, and 2) adaptive, in which pupils work by themselves under a specific skill, contained under the topic of addition and subtraction.
The dataset that was available for this modeling contained a total of 53M interactions of 50k pupils on 190K exercises.

The dataset contains logged interactions in the form of an MDP. Each row in the dataset is defined as a tuple of the form $\{s, a, r, s', d\}$, as defined under section 3.1. In the next section we discuss exactly how the MDP is defined under the KT model.

3.2.3 Dataset Preprocessing

Matrix factorization model KT KT is a knowledge-tracing model used in the company's learning platform. KT is a matrix factorization model that traces a pupil's knowledge of a certain subject and additionally embeds exercises in a vector space according to their difficulty. KT was trained under the subject of mathematics. KT is used by the host company, and its purpose is threefold:

- Trace the knowledge of a pupil under a subject (e.g mathematics), by mapping the pupils' knowledge into an arbitrary vector.
- Map exercises of a certain subject (e.g mathematics) into embeddings that express the difficulty of each exercise,
- Combine the two latter vectors to get predict the outcome of a pupil answering a certain exercise, given his knowledge and the difficulty embedding of an exercise.

KT defines a R^{154} continuous positive vector that describes the skills of a user under mathematics. Additionally, KT embeds exercises in a R^{154} continuous space, bringing exercises with similar difficulty closer together.
To apply the dataset in the form of MDPs, (s, a, r, s', d), we preprocessed the dataset and calculated the KT state of a pupil throughout its traces. Remember that KT defines the state and action space space under the MDP, first discussed in section 3.1.
To acquire a pupil's KT states of a pupil in the dataset, The KT user state update formula was used along a pupil's traces. In this way, the KT user state of the pupil at each given point of its answers was calculated, so as to form the experience replay buffer.
KT's user state update formula is defined as:

$$KTstate_t = reLU(KTstate_{t-1} + 0.01(c - P_{KT}(c = 1|s, a) * a)) \tag{27}$$

where KT_{t-1} is the previous KT pupil state, a is the continuous embedding of the exercise given to the pupil trained under the KT model $\mathbb{R}^{154}$, c is the response of the pupil where $c \in \{0, 1\}$, and $P(c = 1|S_t, a)$ is the solve probability of the pupil's response = 1 under the KT model. Equation 27 is wrapped around the reLU function where $reLU = max(0, x)$.
$P(c = 1|S_t, a)$ is the defined solve probability of a pupil in answering a given exercise. It is defined as:

$$P_{KT}(c = 1|S_t, a_t) = sigmoid(S_t \cdot a_t^T + b_{a_t}^T) \tag{28}$$

where b is the corresponding bias associated with the exercise embedding a. To provide a solve probability in the range of [0, 1], the equation is wrapped around a sigmoid function where $sigmoid = \frac{1}{1+e^{-x}}$.

Reward Function Part of preprocessing the dataset meant that there was a need to define and obtain a reward along the logged pupil traces. To do this, each pupil's traces were located across the dataset. Subsequently, the reward for each interaction was defined as the Delta Average Solve Probability (DASP) under a skill.
In our case, DASP is defined under a specific skill in the topic of addition and subtraction in mathematics. First, we define the Average Solve Probability (ASP) under a skill as:

$$ASP_i = \frac{1}{N}\sum_{i=0}^{N} P(c = 1|S_t, a_i) \tag{29}$$

Where $\sum_{i=0}^{N} P(r = 1|S_t, a_i)$ is the sum of the solve probabilities of all N exercises under this skill. $P(c = 1|S_t, a_i)$ is calculated as defined in equation 28.
To get the reward DASP, we simply subtract the ASP of the previous timestep t-1 over the current ASP:

$$r = ASP_t - ASP_{t-1} \tag{30}$$

The reward function then formulates the growth of the pupil under a specific skill in addition and subtraction. The largest skill under the topic of addition and subtraction was chosen; this skill contained a total of 5k exercises.

3.2.4 Comparing Between Horizons

In this thesis, we aim to compare models that look ahead on a different horizon. To this end, we aim to modify the horizon of the RL models by manipulating the dataset's episode terminals. Two RL models will be created: The first will be called RL-10, in which the dataset will have every 10th exercise answered by the pupil set as an episode terminal. The second variant will be called RL-20, in which every 20th exercise answered by a student will be set as an episode terminal. By modifying the Replay buffer in this way, each model will have a different look ahead horizon for getting the optimal returns.
Figure 5 highlights how the dataset was manipulated in order to create RL models with different look ahead horizons.

3.3 TD3-BC

In this section, we discuss the RL architecture used in all experiments of this thesis. TD3-BC was used for offline RL training in logged data of the behavioral policy BP. We fit TD3-BC on the logged data from BP to improve upon BP's behavioral policy π_β.
TD3-BC is an extension of the TD3 model discussed in Section 2.2.2. TD3-BC modifies the policy update step of regular TD3 by adding a behavioral cloning regularization term. This forces the model to favor actions that are contained in the dataset [42] and minimizes distributional shift.

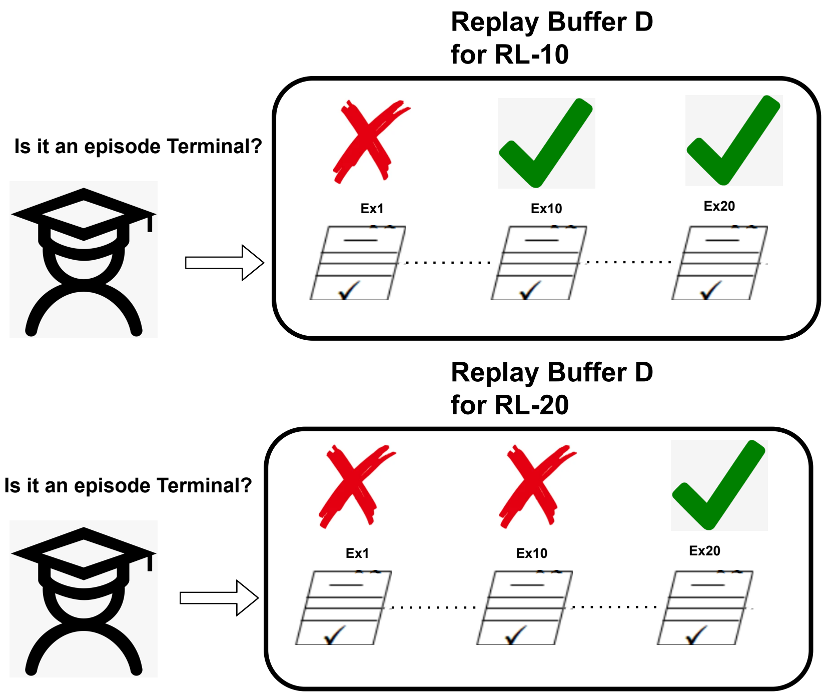
Replay Buffer D for RL-10
Is it an episode Terminal?
Ex1
Ex10
Ex20
Replay Buffer D for RL-20
Is it an episode Terminal?
Ex1
Ex10
Ex20

Figure 5: *Difference in the dataset/replay buffer between RL-10 and RL-20. In RL-10, In the sequence of logged traces across each pupil, each 10nth exercise is marked as the end of an episode. For RL-20, each 20nth exercise is marked as the end of an episode. This allows each model to look ahead for a different horizon, in order to achieve the maximum amount of cumulative returns.*

The new policy update step is defined as

$$\underset{\theta}{max} \underset{s,a\sim D}{\mathbb{E}}[-Q_{\phi 1}(s,\pi_\theta(s)) + (\pi_\theta(s) - a)^2] \tag{31}$$

, where λ is a behavioral cloning regularization term defined as

$$\lambda = \frac{a}{\frac{1}{N}\sum_{s_i,a_i}|Q(s_i,a_i)|} \tag{32}$$

In λ, the denominator contains a simple normalization term over the average Q value of the mini-batch during the policy update. α is a new hyperparameter on top of TD3, used to control the behavioral cloning regularization strength. The smaller the α, the more the agent favors actions that are contained in the dataset. In contrast, the bigger the α, the agent tends to perform less imitation and acts more like an RL agent.

Adding the behavior cloning regularization term 32 on the policy update step of the algorithm in equation 31 is virtually the only change the author implemented on the TD3 algorithm.

TD3-BC's training procedure is shown on the pseudocode below.

Algorithm 2 TD3-BC

```
1: Input: Initial policy parameters $\theta$, Q-function parameters $\phi_1, \phi_2$, replay buffer D with logged data from behavioural policy $\pi_\beta$
2: Input: Initialize $\alpha$ for behavioral cloning regularization term $\lambda$
3: Set target parameters equal to main parameters $\theta_{targ} \rightarrow \theta$, $\phi_{targ,1} \rightarrow \phi_1$, , $\phi_{targ,2} \rightarrow \phi_2$.
4: for j in range(however many updates) do
5:     Randomly sample a batch of transitions, $B = \{(s, a, r, s', d\}$ from D
6:     Compute target actions
           $a'(s') = clip(\mu_{\theta_{targ}}(s') + clip(\varepsilon, -c, c, a_{Low}, a_{High}), \varepsilon \sim N(0, \sigma)$
7:     Compute targets
           $y = r + \gamma(1-d) \underset{i=1,2}{min} Q_{\phi targ,i}(s', a'(s'))$
8:     update Q-functions by one step of gradient descent:
           $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi i}(s,a) - y)^2, i = 1,2$
9:     if j mod policy delay = 0 then
10:        Update policy with gradient ascent:
               $\nabla_\theta \frac{1}{|B|} \sum_{s,a \in B} (-\lambda Q_{\phi 1}(s, \pi_\theta(s)) + (\pi_\theta(s) - a)^2)$
11:        Update target networks with
               $\phi_{targ,i} \leftarrow \rho \phi_{targ,i} + (1-\rho)\phi, i = 1,2$
               $\theta_{targ} \leftarrow \rho \theta_{targ} + (1-\rho)$
12:    end if
13: end for
```

3.4 Simulation

In this section, The implemented simulation will be discussed. This simulation aims to imitate the adaptive feature of the platform, in which pupils are served exercises without the guidance of a teacher. The purpose of the simulation is twofold: First, we use the simulator to evaluate our trained RL agents during training time, in order to estimate their induced growth in pupils. Second, we run the RL-10 and RL-20 variants in the simulator and compare their induced pupil growth against BP's historically induced growth. This is done so as to answer research questions 1 and 2.

Simulations in RL are used to train and evaluate an RL agent. Papers on the frontier of RL use benchmarked simulators to evaluate the RL agent's performance and develop new RL algorithms [36, 53]. This is done as 1) simulators act as a restricted environment in which new algorithms can safely be developed, 2) benchmarked methods make it easier to compare RL methods against currently existing state-of-the-art methods, and 3) They provide a safe alternative to deploying models, especially when deploying underperforming models in the real world can cause considerable harm.

The simulator implemented in this thesis aims to simulate the company's adaptive learning platform, in which pupils work by themselves in the classroom. The pupils answer exercises that aim to maximize their skills under a specific topic, in our case addition and subtraction. To simulate the adaptive learning platform of the company, we need to simulate the pupil's state under KT at a given time point, as well as model the pupil itself to give a response to a proposed exercise. Figure 6 provides an overview of the simulator and its different components which will be discussed in detail in the upcoming paragraphs.

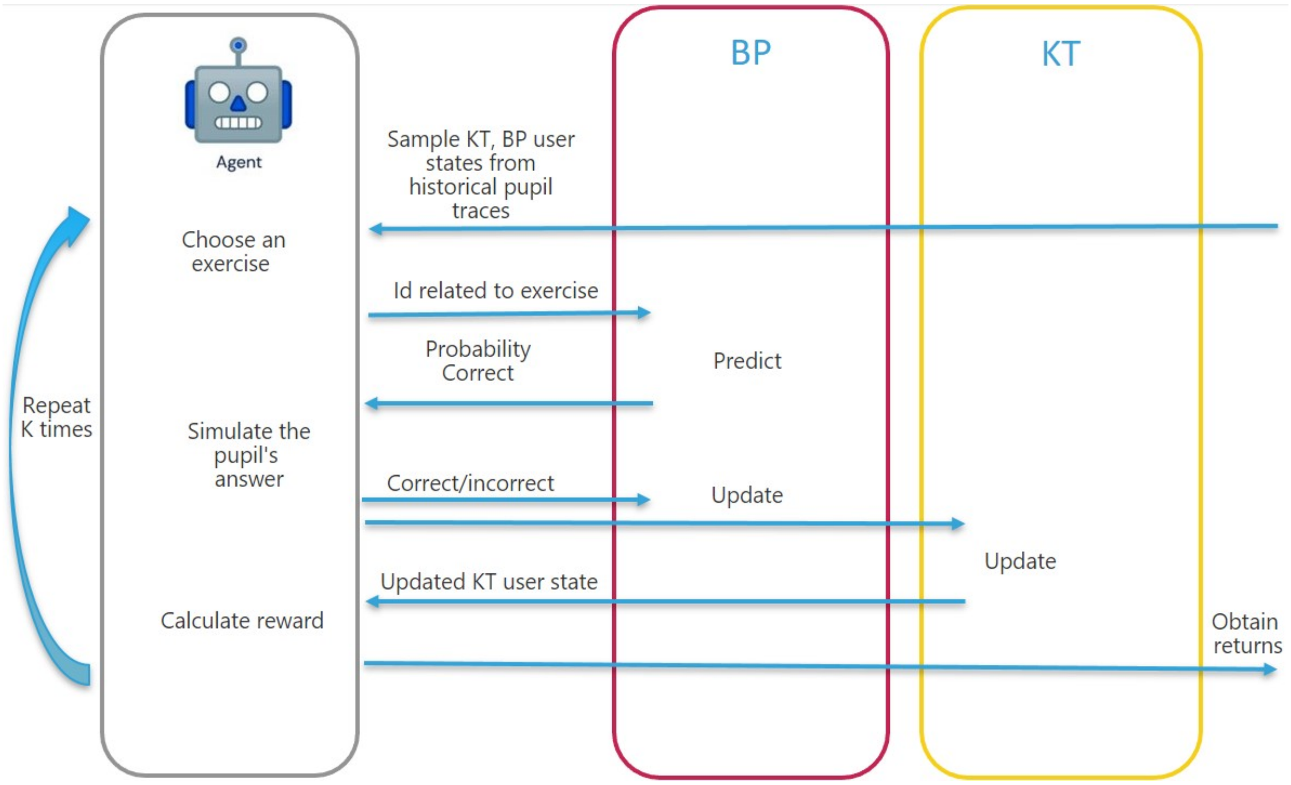

Figure 6: *Overview of the implemented simulator: The simulator aims to imitate the adaptive learning feature of the company's platform, in which pupils are served exercises to work individually without the guidance of a teacher.*

3.4.1 Initializing the simulator

During model evaluation on the simulator, there is a need to simulate the adaptive learning platform as precisely as possible. One of the steps taken for this is the following: When an episode on the simulator is initialized, the pupil's KT and BP should be randomly initialized from the historical data. In this way, we simulate a pupil's adaptive session at a random point in time. The aim of the next paragraphs is to discuss the process of sampling these KT, and BP states.
The logged data on which the TD3-BC model is fit is obtained from student traces over a two-year time period. From this two-year period, the simulator is initialized by uniformly choosing a pupil state s from KT and BP state n. The states are sampled on a point where the pupils were using the 1) adaptive section of the platform, 2) after at least 400 exercises had been answered in the topic of addition and subtraction, 3) the pupils were answering an exercise under a specific skill of addition and subtraction. This ensured that: the BP and KT models were sufficiently warmed up for a given pupil, and that the pupils were working on relevant information.

KT and BP traces For the simulator to work, starting each episode the simulator samples the KT and BP states of the pupil at a given time point. The KT state defines the user state at a given point, while the BP state is used to calculate the solve probability of a pupil answering an exercise, and therefore simulate a pupil's answer in the platform.
The simulator samples the KT and BP states out of a pool of eligible traces. This pool was collected by a subset of 2000 randomly chosen pupils observed historically in the dataset. For these randomly chosen pupils, their underlying BP and KT traces were calculated. A sample of 2000 pupils was selected as it was computationally costly to calculate the traces for all 47k pupils. This resulted in a total trace pool of shape of 17515 rows, with each row containing an equivalent KT and BP state for the pupil at a given time point. The fact that we only contain 17515 state traces across 2000 pupils is attributed to the fact that we picked traces of pupils that were working under a specific skill at the time. The topic of addition and subtraction contains 81 different skills, effectively minimizing the pool of traces by a significant amount.

3.4.2 Wolpertinger Architecture for Discretizing Continuous Actions

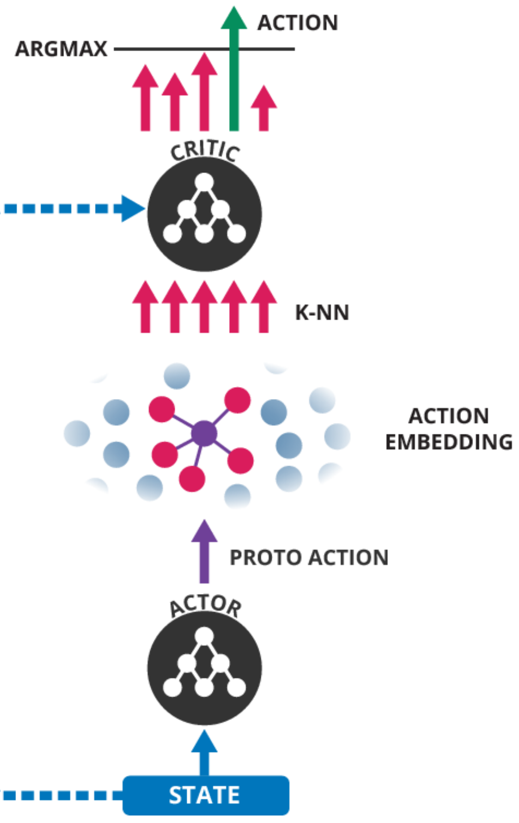

Figure 7: *Image outlining the process in which the Wolpertinger architecture chooses an action. First, a state is sampled from the simulator's environment. The actor chooses a proto-action that will most likely not lie in the valid exercise space. The K nearest neighbors to the proto-action are calculated and are each evaluated through the critic. Finally, an exercise in the valid exercise range is chosen, which contains the highest Q-value. Image credits: [52].*

In this thesis, we use TD3-BC, an offline RL variant of TD3 discussed in paragraph 3.3. Remember that TD3 only works on continuous action spaces, and in our RL problem formulation, the MDP's action space is defined in a closed set A of continuous vectors under the KT model, $A = [a_1, a_2 ..{}_n] \in$

R^{154}. In the simulation, when the RL agent maps a state into a continuous action, it should be able to discretize the continuous proto-actions by selecting a discrete exercise. The policy of the RL agent should then map a continuous proto-action $\hat{a} \notin A$ into a continuous embedded vector $a \in A$, where each embedded vector represents a discrete exercise. In this case, the nearest embedded vector $a \in A$ is chosen if it has the highest Q-value compared to its neighbors.
Discretizing a continuous action by the RL agent into a concrete exercise is done by implementing the Wolpertinger architecture [52]. Figure 7 shows the process in which the Wolpertinger architecture chooses an action.
The Wolpertinger architecture works as follows: first, the actor maps a state into a continuous proto-action $\hat{a}$ where the proto-action is unlikely to be part of an exercise set, $\hat{a} \notin A$. Then, the K nearest neighbors closest to $\hat{a}$ are located. Finally, the critic iterates through all state-action pairs (s,a) from the K nearest actions and selects the action with the highest Q-value.

Algorithm 3 Wolpertinger Policy

```
State s previously received from the environment
$\hat{a} = f^{\pi}_{\theta}$(s) { Receive proto-action from actor }
$A_k = g_k(\hat{a})$ {Retrieve k approximately closest actions}
$a = argmax_{\{a_j \in A_k\}} Q^{Q}_{\theta}(s, a_j)$ apply a to environment; receive r,s'.
```

Embedding the exercises into a set of continuous vectors allows the model to leverage information about the difficulty of each exercise itself. The policy in turn learns to generalize and scale into huge action spaces, which is allowed by selecting a ”neighborhood” of exercises, in which the exercise with the maximum Q-value exists [52]. In this way, we overcome problems that exist over formulating an RL agent in big discrete action spaces, like the computational intensity of evaluating all actions. We also avoid Q-function approximators with a huge output dimensionality, which tend to underperform compared to RL agents with a smaller output dimensionality [54].

Exercise Exhaustion In the simulation, during an episode, an exercise cannot be selected more than once. This is to enforce constraints related to the company's platform. Therefore, when an exercise is selected, this exercise is exhausted and cannot be selected for the pupil again during the ongoing episode. Once the simulator has been reset, All exercises become available for selection again.

3.4.3 Simulating the Pupil's Response

For the simulation, one important aspect to consider is to simulate a pupil's response to a suggested exercise. For the simulator to be accurate, there is the need of finding a model which can approximate the pupil's answer accurately enough, as if they are answering on the learning platform. To this end, we used the myopic BP, the same model that served exercises in the dataset, to simulate the response of a pupil to a given exercise: The BP outputs a solve probability $P_{BP}(c = 1|s_n)$, which indicates the solve probability of a pupil answering an exercise correctly, given the BP state. The answer is then realized with a biased coin flip.

3.4.4 Simulation Episode Example

We conclude this section by providing a pseudocode that outlines an example of an episode inside of the simulation.

Algorithm 4 Simulation episode

```
1: Input: randomly sampled user state s and BP state n from a pool of eligible traces
2: Initialize total cumulative returns: $R \leftarrow 0$
3: Initialize the number of steps K to take inside an episode
4: for K steps do
5:     TD3-BC agent retrieves proto-action $\hat{a}$
6:     Apply Wolpertinger to $\hat{a}$, get exercise a
7:     BP retrieves a, calculates $P_{BP}(c=1|n)$
8:     Simulate the pupil answer by realizing $P_{BP}(c=1|n)$ into response c $\in \{0,1\}$
9:     Update the BP state given pupil's response c
10:    Update the user state s by:
           $$s \leftarrow reLU(s + 0.01(c - P_{KT}(c=1|s,a) * a))$$
11:    Observe reward r (DASP) by:
           $$r \leftarrow \frac{1}{N}\sum_{i=0}^{N} P_{KT}(c=1|s|a_i) - \frac{1}{N}\sum_{i=0}^{N} P_{KT}(c=1|s_{previous}|a_i)$$
12: end for
```

Task	Subject	Topic	Skills	# Exercises
1	Maths	Addition & subtraction	All skills	100k
2	Maths	Addition & subtraction	Addition & subtraction up to 20 by heart	5k

Table 1: Table describing the exercise space in the simulator for each of the two tasks. For task 1, the simulator is ”emulating” the host company’s adaptive platform under the topic of addition and subtraction. In the task, the RL agent can choose between 100k different exercises to serve the pupil. Since historically the BP served exercises under only specific skills, in task 2 we restrict the exercise space in which the RL agent can choose from only under the skill of ”Addition and subtraction up to 20 by heart”.

4 Experimental Setup

This section describes the experiments conducted in this thesis, in order to answer the research questions proposed in paragraph 1.3. To do this, two different tasks were performed: For the first task, we aim to create and train two RL variants, RL-10 and RL-20. These models will be trained offline on the dataset that contains the logged interactions from BP. Each RL model is evaluated in the simulator in order to estimate their performance.

For the second task, we pick the best-performing policies for RL-10 and RL-20 and compare them against each other and to the myopic BP. This experiment allows us to compare three different models with different look-ahead horizons (1 vs 10 vs 20) in order to calculate each model’s induced growth in pupils.

4.1 Task 1: Training and fine-tuning RL-10 and RL-20

For Task 1, we first give an overview of the task’s process. We then split down the training and evaluation procedure and go into detail on each procedure. Then, we discuss the grid search we performed to obtain the best model for each RL variant. We also discuss relevant hyperparameters on the grid search space, as well as why we chose them. As the last step, we discuss the actor and critic architecture of the TD3-BC variants as well as the metrics used in this task.

4.1.1 Training and Evaluation Procedure

For the first task, we formalize the RL-10 and RL-20 TD3-BC variants. The variants are formalized by manipulating the episode terminals of the replay buffer as discussed in Section 3.2.4. The aim of this task is to train the models on the historical data and obtain the best-performing policy for both RL-10 and RL-20 variants. In a given experiment, a policy is trained offline for a fixed number of training steps. Then, the policy is evaluated on the simulator to gather its cumulative returns. The procedure is repeated until training converges, and we acquire the optimal policy. Figure 8 outlines how an experiment works for training an RL policy.

Task 1 consists of steps between training the policies by means of gradient descent and evaluating the policy on the simulator in order to evaluate its performance. We first discuss training on the dataset. We then discuss the policy evaluation step on the simulator.

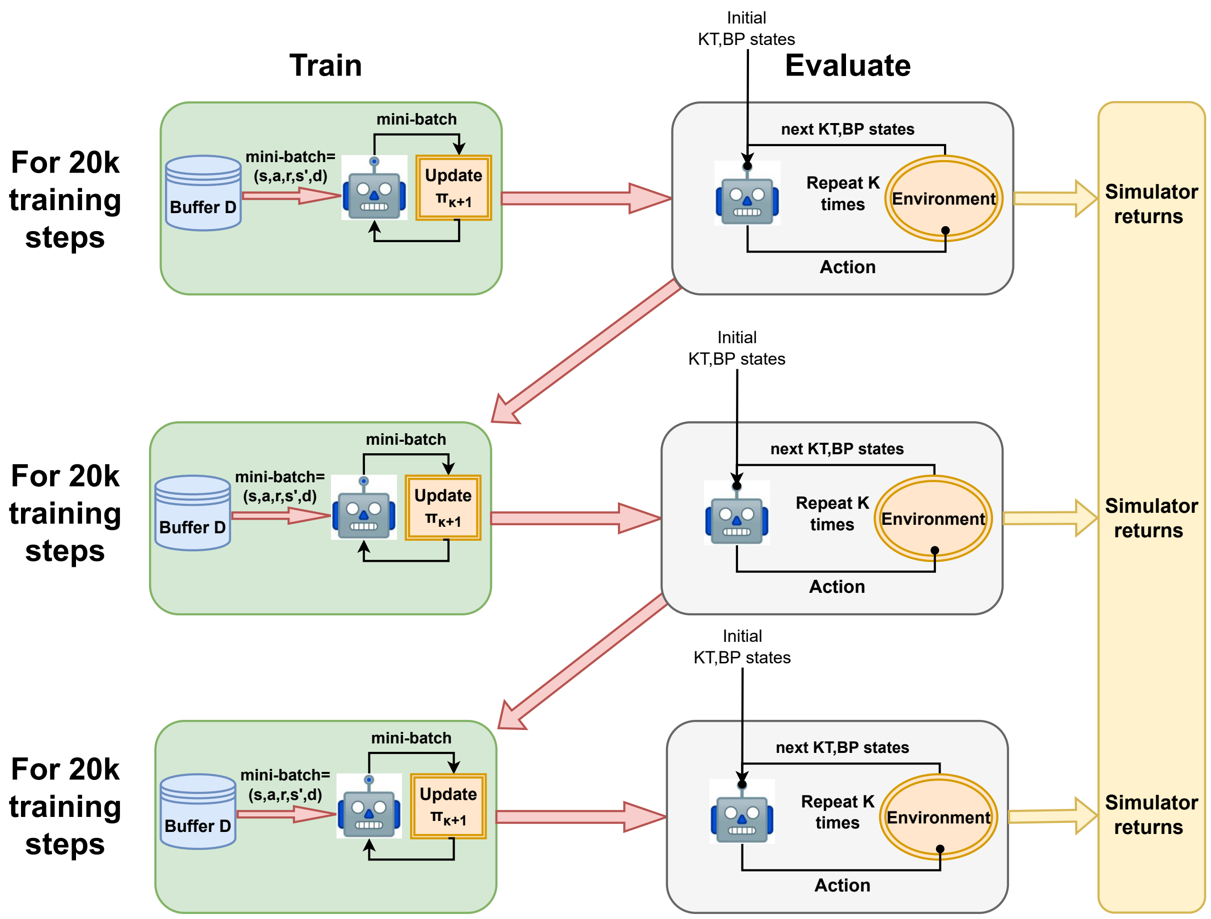

Figure 8: *Overview of task 1: The model is trained by means of mini-batch training for a number of training steps. Then training temporarily stops and the model is run through the simulator so as to gather the cumulative returns. Training resumes, and the process is repeated.*

Training For training the models, offline training on the replay buffer was performed. After every 20k fixed training steps, training stopped and the model was evaluated on the simulator for 5 episodes to gather its estimated returns. Afterward, training resumed for the next 20k steps. Each experiment was run for a total of 300k training steps. If at any point the model's Q-loss inflated to unrealistic values, early stopping was implemented and the experiment stopped. Here we refer to Q-loss as the critic loss of the TD3-BC model, referred to in 20. This problem directly reflects the issue of distributional shift discussed in section 2.3.2: Q-values take a "sharp" incorrect peak for some actions, and the model "visits" states that are unseen in the dataset. Eventually, inflated Q-values cause failure mode.

During training, we normalize the features of every state in the replay buffer. We also normalize the rewards observed in the replay buffer. Normalizing both the rewards and the features of each state provide better convergence properties for non-linear function approximators like neural networks [55] [56].

Each model was trained on the whole dataset defined in 3.2.2. This means that in all experiments 1) models were fit into a replay buffer of 53 million interactions, 2) under the topic of addition and subtraction, and 3) with interactions both in adaptive and lesson. Each RL-10 and RL-20's DASP reward was defined under the biggest skill in the topic of addition and subtraction, named "Addition and subtraction problems up to 20 by heart". The models, therefore, aimed to induce a policy that maximizes growth in the underlying skill.

Evaluation For each policy evaluation step on the simulator, we run the trained policy on the simulator for 5 different episodes. Since each RL model is trained to optimize returns across a certain number of steps, for RL-10, episodes in the simulator will consist of a total of 10 steps of exercise suggestion. Likewise for RL-20, each episode in the simulator will consist of 20 steps. comparing across different steps makes it invalid to compare the RL-10 and RL-20 variants. Table 1 describes the eligible exercises that each RL variant can choose in each task in the simulator. Specifically, the models were able to choose 100k different exercises under the whole topic of addition and subtraction. For all experiments the Wolpertinger ratio was set to 0.05, meaning that the agent chose the 5000 nearest neighbors of the actor's proto-action for exercise selection.
Evaluating the policies over the whole topic of addition and subtraction was feasible as the Wolpertinger architecture allows us to scale into huge action spaces. Also, each policy was trained on past interactions over the whole topic, which means that we can create a policy that serves exercises over this whole topic. On Task 2, we restrict the eligible action space over a specific skill of ”addition and subtraction up to 20 by heart”.

Grid-search To obtain the best-performing model for each RL variant, grid-search across a hyperparameter space was performed. The best-performing model on the simulator was picked and then used for the second task which will be discussed later. First, some grid searches were performed to find the most important hyperparameters to tune for the TD3-BC models. We will be only reporting the final grid search performed for this task. The final grid search performed across both RL-10 and RL-20 variants is reported in Table **??**.
Below, we briefly explain the function of each hyperparameter.

- Evaluation frequency: After how many training steps to start policy evaluation on the simulator
- Total training steps: Total number of training steps to train a policy
- Discount: Discount factor gamma for critic training
- Tau: interpolation factor τ for Polyak averaging
- Policy noise: Controls the noise injected into the policy during training time for regularization.
- Noise clip: upper and lower range of the noise
- Policy update frequency: After how many training steps should a policy update step occur
- alpha: TD3-BC's behavioral regularization term α.
- K-ratio: ratio of nearest neighbors to choose from, over the total exercise space for the Wolpertinger architecture.

The focus of the grid search was on the actor and critic learning rate. Initial grid searches showed that low learning rates stabilized the models but made training convergence very slow. Grid search also focused on α as it is an important hyperparameter for offline RL [42]. Specifically, α controls the trade-off between regular RL and behavioral cloning. The lower the α, the more the model outputs actions close to the ones observed in the dataset and therefore mitigates distributional shift. On the other hand, the bigger the α, the more the model acts as an RL model, but the bigger the risk of

	Hyperparameters	Values
Experiment	**RL-variant**	[10, 20]
	Evaluation frequency	20k
	Total training steps	300k
	Normalize states	True
	Normalize Rewards	True
Architecture	Critic Hidden dim	2
	Critic hidden layers	256
	Critic activation function	RELU
	Actor hidden dim	2
	Actor hidden layers	256
	Actor activation function	RELU
	Actor + critic learning rate	$[3*10^{-3}, 3*10^{-4}]$
TD3	Batch size	256
	Discount	0.90
	Tau	0.005
	Policy noise	0.09
	Noise clip	0.2
	Policy update frequency	100
TD3+BC	**alpha**	[1.3, 1.7, 2.5]
Wolpertinger	K-ratio	0.05

Table 2: Grid search performed across both RL-10 and RL-20 variants. Number's that are in brackets indicate the hyperparameter space in which grid search wil lbe performed on. All other hyperparameters are kept constant across the grid search. Parameters in bold denote the parameters that will be explored during the grid-search.

distributional shift [42]. The choice of α is in our case more conservative. While TD3-BC's original author implementation kept α at 2.5, our models suffered from distributional shift when $alpha$ was set to > 2.5. Therefore, α was kept lower to make the models act more conservatively [42]. As the logged BP can be considered an ”expert demonstrator”, it makes sense to stay close to its behavior.
It was also found out that the models destabilize quickly due to exploding loss of the critic. The author's implementation was originally set to 2, but in our case training quickly destabilized and failed to converge to an optimum. In initial training runs, the policy update frequency was set to 10, 20, and 50. We found that these values did not help the model converge, and the critic loss exploded nonetheless. In the end, the policy update frequency was set to 100 which provided better-converging properties [36].

Actor and critic architecture For all experiments, we used a 2 hidden layers architecture for both actor and critic, with a hidden layer size of 256 each. All layers of the actor and critic architectures

were fully connected. These settings were kept the same as per the TD3-BC author's implementation of the model [42]. We used reLU as the activation function for both actor and critic, in line with the KT model described in 3.2.3. We also chose reLU for the actor, as the exercise embeddings of the KT model were also trained with a reLU activation function. As per the author's implementation, actor and critic networks use the Adam optimizer for training [42, 57].
The input and output dimensionality of the critic and actor are (308, 1) and (154, 154) respectively.

Evaluation and metrics To evaluate the best-performing policy across each RL variant, We pick the best-performing policy on the last evaluation step in the simulator. The metric used for this is the average returns on the simulator as well as its standard deviation. As mentioned in paragraph 2.1, RL models aim to solve MDPs by maximizing the expected returns across an episode, which is reflected by this metric. We evaluate the models by the Average Returns Across Episodes (ARAE):

$$ARAE = \frac{1}{n}\sum_{j=0}^{n}\sum_{i=0}^{k} r_k \tag{33}$$

where k reflects the number of steps in the simulator for each RL-variant, and n is the number of episodes that are being averaged in each evaluation step. In our case for the experiments, for RL-10 k is set to 10, and for RL-20 k is set to 20. We evaluate over 5 episodes for each evaluation procedure, therefore n = 5. The ARAE is normalized according to the mean and standard deviations of the rewards observed in the dataset from which we trained the RL models.

Policy evaluation step example The pseudocode in the next page explains the training and evaluation procedure taken in order to train the RL-10 and RL-20 variants. Specifically, it focuses on the steps taken in a given episode in the simulator in order to evaluate each model.

4.2 Task 2: Comparing Different Horizon Variants Against The Simulator

In this task, we aim to answer the research questions defined in paragraph 1.3. To do this, we compare the growth induced in pupils by three different models: 1) the BP, which is myopic, RL-10 2), which looks to maximize pupil growth in the next 10 steps, and 3) RL-20, which looks to maximize pupil growth in the next 20 steps. We aim to compare the induced growth in pupils by three models with different look ahead horizons. We take the best-performing RL-10 and RL-20 models from task 1 and use them for the experiments conducted for this task.
As initially discussed in Section 3.2.2, in the adaptive learning feature of the host company's platform, the pupils answer questions under a specific skill. To this end, we execute task 2 by first gathering all 90k sessions from the historical data where pupils were being served exercises under a specific skill in adaptive. For these sessions, we calculated the equivalent growth induced by the BP. As a next step, we take the pupil state at the beginning of each adaptive session and run it through the simulator to obtain the growth induced by the RL-10 and RL-20 models. For the simulator, we only sample 12k of these sessions, as sampling all 90k adaptive sessions would be computationally expensive. We ran the traces for different lengths of K steps in the simulator in order to compare them against the BP baseline.
First, we discuss adaptive sessions. Then we discuss how we gathered the adaptive traces of each adaptive session. By adaptive traces we mean the beginning point of each adaptive session observed in the dataset. Afterward, we discuss the hyperparameter configuration for the RL models and the metric used to calculate growth for each model.

Algorithm 5 Task 1: Training and evaluating the RL models

```
1: Input: total number of training steps
2: Input: pool of BP , KT states
3: Initialize number of steps K to take inside an episode
4: Input: Initial actor and critic parameters for TD3-BC training
5: for j in range(however many updates) do
6:     Randomly sample a batch of transitions, $B = \{(s,a,r,s',d\}$ from D
7:     Train TD3-BC agent by means of gradient descent
8:     if its time to evaluate on simulator then
9:         for K episodes do
10:            $R \rightarrow 0$
11:            Uniformly sample $s_{BP}$, $s_{KT}$ pair of states
12:            for K steps do
13:                TD3-BC agent retrieves proto-action $\hat{a}$
14:                Apply wolpertinger to $\hat{a}$, get exercise a
15:                BP retrieves a, calculates $P_{BP}(c = 1|n)$
16:                Simulate the pupil answer by realizing $P_{BP}(c = 1|n)$ into response c $\in \{0,1\}$
17:                Update the BP state given pupil's response c
18:                Update the user state s by:
                       $s \leftarrow reLU(s + 0.01(c - P_{KT}(c = 1|s_{KT}, a) * a))$
19:                Observe reward r by:
                       $r \leftarrow \frac{1}{N}\sum_{i=0}^{N} P_{KT}(c = 1|s_{KT}, a_i) - \frac{1}{N}\sum_{i=0}^{N} P_{KT}(c = 1|s_{KTprevious}, a_i)$
20:                Add reward to total cumulative returns:
                       $R \leftarrow R + r$
21:            end for
22:            Store R
23:            Resume training until early stopping or until end of loop
24:        end for
```

Adaptive sessions The historical data used to train the RL agents contain adaptive sessions of pupils. An adaptive session consists of a pupil's session in the adaptive feature of the platform, where they answer exercises under a specific skill of addition and subtraction. Adaptive sessions end when a pupil starts working on a different topic or skill.
For this task, we calculate growth only under a specific skill which the pupils were working on historically. This skill is the same one used for shaping the trained RL models' reward in task 1. The skill's name is "Addition and subtraction problems up to 20 by heart".

Experiment configuration In order to conduct this experiment, the adaptive sessions of pupils were gathered in a specific skill on the topic of addition and subtraction. From these sessions we obtained 1) the growth of each adaptive session of a pupil, used to calculate the average growth for each model, 2) the respective length of each adaptive session (how many exercises a pupil answered on the go), 3) and the beginning BP and KT traces of each session. The beginning traces were run through the simulator for the respective RL-10 and RL-20 models so as to obtain the respective growths over the same amount of steps in the historical adaptive sessions.

We only obtain adaptive sessions from the skill under which the reward of the RL models was shaped, namely ”Addition and subtraction problems up to 20 by heart”. A total of 90k adaptive sessions were observed in this skill, with the length of sessions ranging from 1 to 900. For costs and computational reasons, we could not run all 90000 observed historical traces through the simulator. Therefore, traces were sampled from a similar beginning distribution, with a total of 11600 traces being sampled. Adaptive traces with a session length of 1 and up to 40 steps were sampled for this task. Figure 9 compares the distribution of adaptive sessions observed in the dataset and the distribution of sampled sessions that were ran in the simulator.

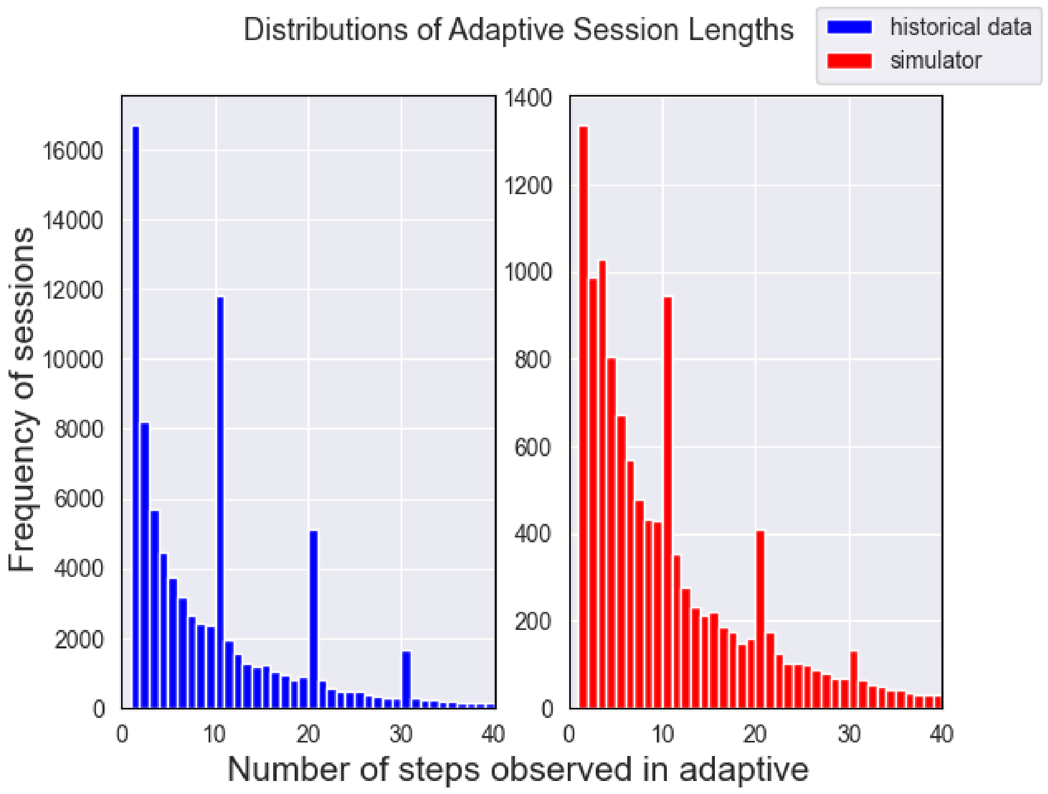

Figure 9: Comparison between distributions. Left: distribution of adaptive session frequency observed in the historical data, with the X axis being the number of steps in each adaptive session. Right: distribution of adaptive sessions frequency sampled to run through the simulator. The resulting distribution of the simulator was slightly more skewed to the left so as to emphasise sampling adaptive sessions of bigger lengths.

Hyperparameter configuration We chose the best-performing models from task 1, for both RL-10 and RL-20. Choosing the best performing models was based on Figures 10 and 11. The chosen models had the hyperparameter configuration denoted in Table **??**.

	Hyperparameters	Values
Architecture	Critic Hidden dim	2
	Critic hidden layers	256
	Critic activation function	RELU
	Actor hidden dim	2
	actor hidden layers	256
	Actor activation function	RELU
	Actor + critic learning rate	$3 * 10^{-3}$
TD3	Batch size	256
	Discount	0.90
	Tau	0.005
	Policy noise	0.09
	Noise clip	0.2
	Policy update frequency	100
TD3+BC	alpha	1.7
Wolpertinger	K-ratio	0.2

Table 3: Hyperparameter configuration of the trained models used in task 2. The hyperparameter configuration was taken from the best performing models for each RL variant in Task 1. We shift the wolpertinger K-ratio from 0.05 to 0.2 as the available pool of exercises for this task is much lower.

We keep the hyperparameter configuration the same across both models as 1) this will allow for a more fair comparison and 2) as this configuration converged to the best policy across both variants.

4.2.1 Evaluation and metrics

The evaluation metric used in this experiment is called the $KT\text{-}\overline{DASP}$ over all episodes. Intuitively, this metric aims to capture the average growth observed across all adaptive sessions of the same length, for all pupils. We will use the term growth interchangeably with the term $KT\text{-}\overline{DASP}$.
We will break down the metric into individual parts below. $KT\text{-}\overline{DASP}$ is defined as:

$$KT\text{-}\overline{DASP} = \frac{1}{N}\sum_{j=0}^{N}\sum_{i=0}^{k}\frac{1}{k}DASP_i \tag{34}$$

where $DASP_i$ is the growth observed in the pupil, after handing them an exercise. DASP is averaged across a whole adaptive session of k steps. Then, it is averaged across all N of the adaptive sessions of the same length.
where $DASP_i = ASP_t - ASP_{t-1}$. Notice that $DASP_i$ is defined in the same way as the reward function in paragraph 3.2.3. In more technical terms, $DASP_i$ is defined as the leap in growth or Average Solve Probability as Defined by the KT model. ASP is defined under the KT model as :

$$ASP_i = \frac{1}{M}\sum_{i=0}^{M}P(c=1|S_t,a_i) \tag{35}$$

Where ASP denotes the average solve probabilities under a specific skill, given the exercise served to the pupil and its KT state at time t. Notice that we average solve probabilities of all exercises under a specific skill. For this metric, we use the same skill under which the model is trained on in task 1, namely ”Addition and subtraction problems up to 20 by heart”.

4.3 Tools and Technologies

For both tasks, the TD3-BC architecture was built on top of Pytorch [58]. The simulation was implemented by using Python 3, NumPy, and Pandas [59, 60, 61]. All experiments performed in this thesis were ran on the Amazon Web Services (AWS) Sagemaker platform for distributed cloud computing. Experiments for Task 1, recruited ml.m5.16xlarge instances from the Sagemaker platform. The ml.m5.16xlarge instances are CPU-based instances with 64 CPU processors and 512GB of ram each.

For each model on task 2, ml.m5.8xlarge instances were recruited from the Sagemaker platform. These are CPU-based instances with 32 CPU processors and 256 GB of memory. Task 1 was performed 5 times, setting a random seed every time to control for noise that occurs during training time. The random seeds in task 1 were set to 11,12,13,110,120 for each RL variant’s grid search. The seed was set on the Pytorch, NumPy, Pandas libraries. This means that the seed was set for the RL variants training procedure and the initialization of the BP and KT traces per episode in the simulator.

For Task 1, each experiment was run for an average of 2-3 hours. For task 2, the experiment was run for a total of 13 hours for each RL variant.

5 Results

In this section, we present and discuss the results of all tasks conducted in this thesis. In the first section, we discuss the results of training and evaluating all RL-10 and RL-20 policies. In the second task, we discuss the results of comparing the three models across different adaptive session lengths.

5.1 Results For Task 1

We start by presenting the grid-search results of RL-10 and RL-20 variants. Figure 10 and 11 denote the training curves of every trained RL-10 and RL-20 variant, proposed in section 4.1.1. In these figures, the average normalized cumulative returns across episodes are highlighted, also called normalized ARAE. Normalized ARAE is plotted in relation to the number of training steps. Each plot denotes a different hyperparameter configuration within the hyperparameter space. The results were averaged over 5 different random seeds for both RL-10 and RL-20 grid searches. The shaded area represents the standard deviation across seeds. We implemented early stopping so as to stop models in which performance dropped rapidly in the simulator. Specifically, we implemented early stopping when the critic loss diverged, as discussed in Section 4.1.1. This is why some models stop training before the total number of 300k training steps.

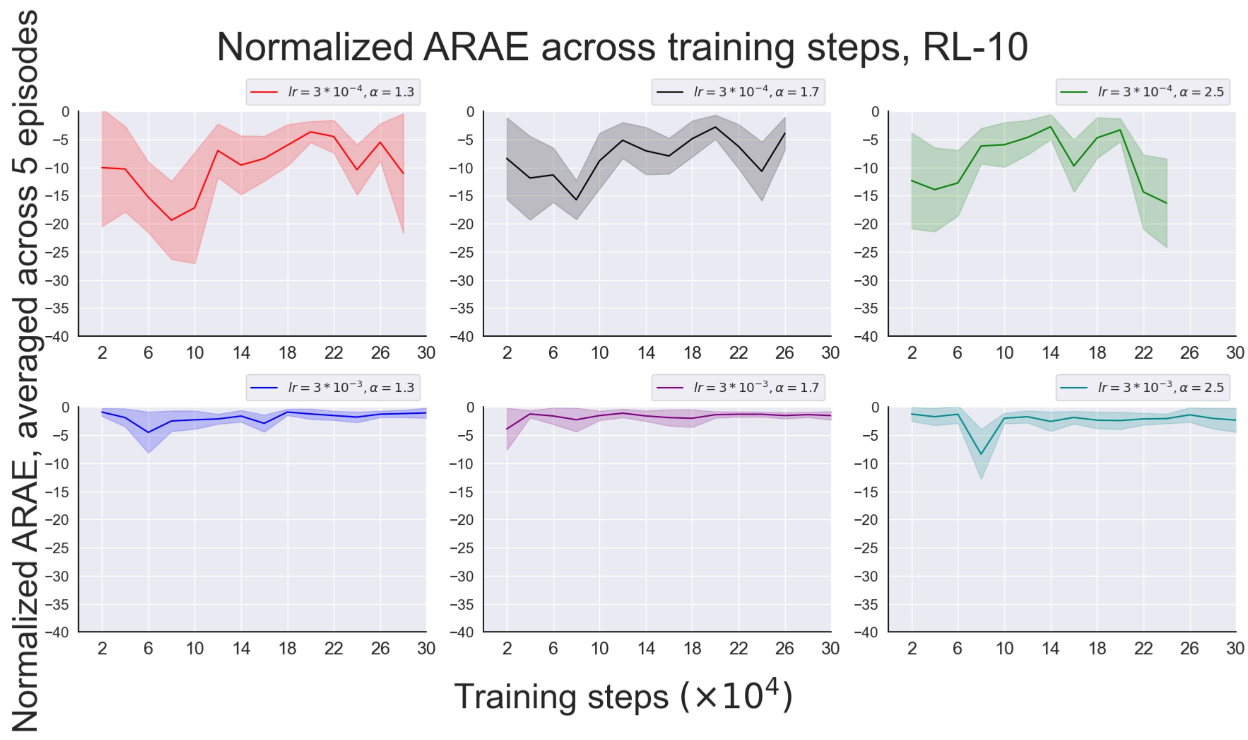

Figure 10: Normalized Average Returns Across Episodes throughout training steps, for the hyperparameter grid search of the RL-10 models. Each plot denotes the results of training an RL-10 model over a certain hyperparameter configuration. lr is the learning rate of the model, while α is the behavioral cloning term regularization of each model. The Y axis denotes the average returns across 5 episodes in a given evaluation step. The X-axis denotes the training steps over time. Results are averaged over 5 random seeds, and the shaded area represents the standard deviation across seeds.

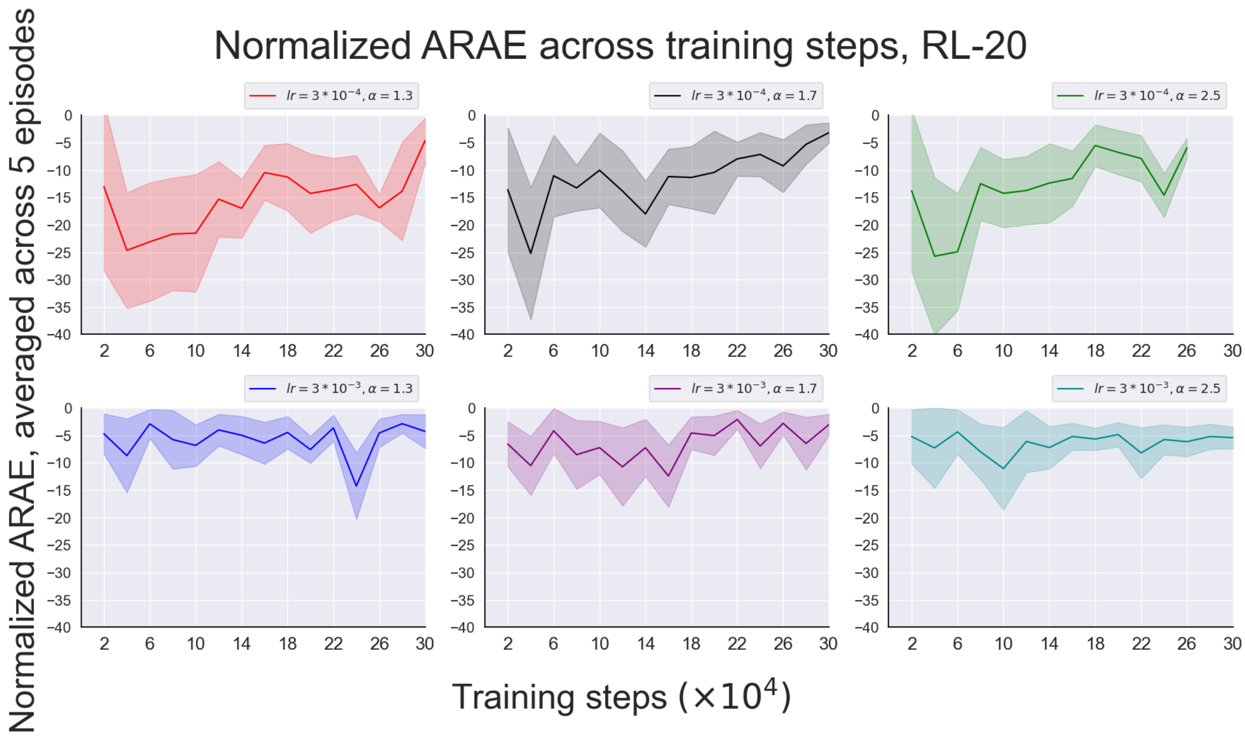

Figure 11: Normalized Average Returns Across Episodes throughout training steps, for the hyper-parameter grid search of the RL-20 models. Results are averaged over 5 random seeds, and the shaded area represents the standard deviation across seeds.

in Figures 10 and 11, training curves for models with a learning rate of $3 * 10^{-3}$ showed fast convergence even in the initial training steps. The higher learning rate of these models helped them converge to a good policy faster than models with a lower learning rate. We observe that a smaller learning rate of $3 * 10^{-4}$ made models in Figures 10 and 11 take too long to converge. In this sense, it is possible that the higher learning rate of $3 * 10^{-3}$ might have gotten models stuck into local minima. The fast convergence of models with a learning rate of $3 * 10^{-3}$ might indicate that during training, the RL models quickly exploit state-action pairs that provide the best cumulative returns in the simulator. It is possible then that the models' actor and critic loss get "stuck" in a mode that makes the actor network provide similar exercises to a pupil across episodes, therefore keeping the normalized ARAE stable across evaluations in the simulator. This might be shown by the relatively static curves produced along the X-axis for RL-10 and RL-20 models with a learning rate of $3 * 10^{-3}$ in Figures 10 and 11.

Given that the behavioral policy in the dataset can be considered an "expert demonstrator", the behavioral cloning regularization term α does not seem to have a big influence on the training curves. We also notice that the learning rate was more important as a hyperparameter compared to α: The models with a learning rate of $3 * 10^{-3}$, but a variable α performed similarly across Figures 10 and 11. Our results are consistent with the ablation studies performed by the authors of TD3-BC, which found that α tends to keep performance robust when kept in the range of $1.5 < \alpha < 4$ [42].

We favored lower α's during hyperparameter search and brought TD3-BC closer towards behavioral cloning because our dataset can be considered an expert demonstrator, or a noisy-expert demonstrator. Noisy-expert demonstrator means that the dataset follows the actions of an expert demonstrator with an occasional shift in optimal actions. While bringing the RL model closer to behavioral cloning is a sensible approach, Kumar claims that expert demonstrators are a necessary but not sufficient condition for preferring behavioral cloning over offline RL algorithms, and the environment structure plays an equally important role [62]. Kumar also claims that offline RL should be preferred over BC when the critical states are few in the environment, with critical states meaning states with only a few eligible actions for the agent to take him towards a region with high positive rewards. In such states, a wrong action might e.g cause the termination of an episode, or lead an agent to get stuck in a region with negative rewards. In contrast, non-critical states have a big volume of actions that can allow them to reach an end goal state [62].

In this sense, it is unclear whether our environment contains a large number of critical states or not. As discussed in the results of task 2, taking a wrong action might cause the agent to observe a proportionally negative reward compared to positive ones during episodes, leading to the observed negative cumulative returns. This might be an indication that the environment structure contains a lot of critical states. Another potential evidence for a large number of critical states is explained in Task 2, where we observe that a more optimal induced growth from RL models to pupils highly depends on the first two exercises served. Seeing that the potential exists for a large amount of critical states in the environment, it would be preferable to keep a lower α for the agent to perform more like behavioral cloning [62].

Looking at Figures 10 and 11, no models managed to obtain positive normalized ARAE in the simulator. It was observed that policies in some episodes accrued positive returns but in general, actions taken by the policies in the simulator usually had a higher negative reward in magnitude compared to the positive rewards, hence the negative normalized ARAE. This is especially prevalent in the negative curves for RL-20 models in Figure 11; because the RL-20 models took 20 steps for each episode in the simulator, they managed to on average gather more negative DASP rewards, which explains the more negative normalized ARAEs observed in Figures 10 and 11. Note that this comparison does not constitute a comparison of the performance between the RL-10 and RL-20 models, because, for each variant, each episode consisted of a different number of steps (10 and 20). The effect of proportionally bigger magnitudes in individual negative rewards compared to positive rewards is also prevalent in Task 2. We explain the cause of this further in paragraph 5.2.

Across both grid searches, the models that performed the best contained a learning rate of $lr = 3 * 10^{-3}$ and $\alpha = 1.7$. The models were chosen according to their Normalized ARAE at the last evaluation step in the simulator. These two models were used for model comparison against the BP for Task 2.

5.2 Results For Task 2

In this section, we discuss the results of the experiment conducted for task 2. We compare across models of three different horizons, namely, BP, RL-10, and RL-20, with horizons of 1, 10, and 20 respectively. We compare the models across different adaptive step lengths. We also compare each model's total induced growth to pupils. The term growth will be used interchangeably with the metric $KT\text{-}\overline{DASP}$.

Figure 12 highlights the $KT\text{-}\overline{DASP}$ plotted across all different adaptive session lengths. Table 4 denotes the total $KT\text{-}\overline{DASP}$ observed across all adaptive sessions for the BP, RL-10, and RL-20 models.

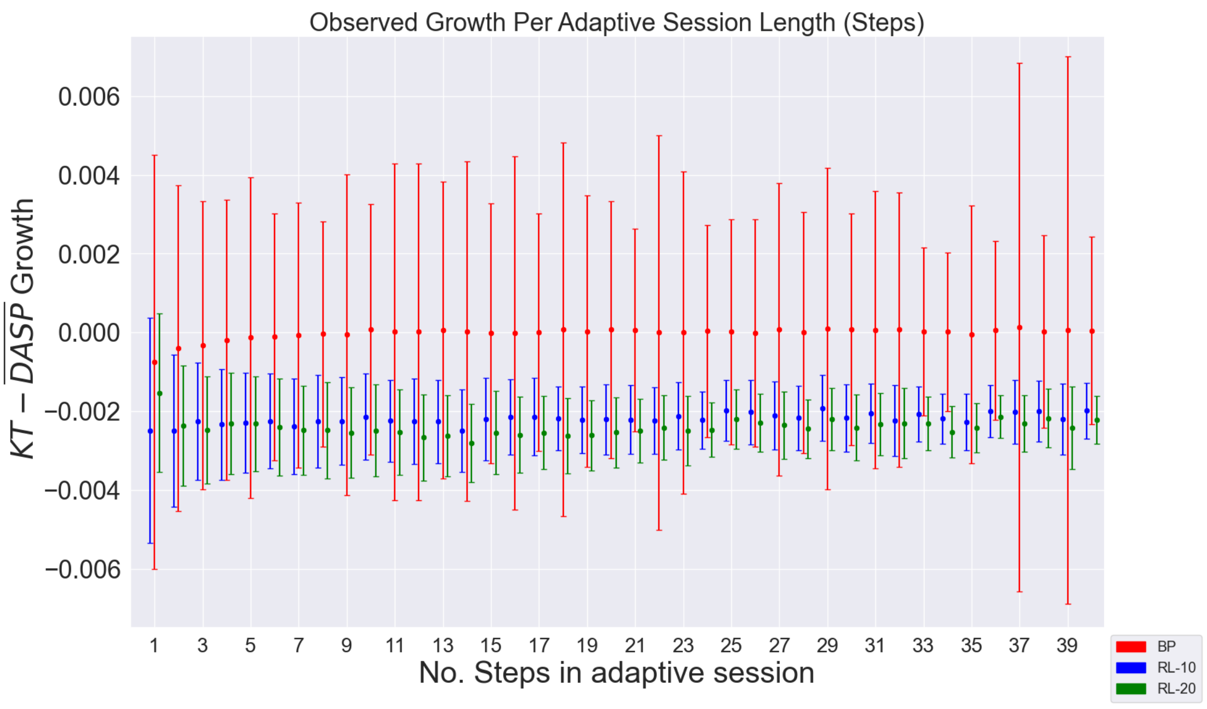

Figure 12: KT-$\overline{DASP}$ growth averaged per adaptive sessions lengths. No. steps in adaptive sessions denote the number of exercises suggested in a given episode, either in the simulator for the RL models or in the historical data for the BP. Lines around the mean represent the standard deviation of the KT-$\overline{DASP}$ across sessions of the same adaptive length.

Model	Total Avg KT-DASP	std
BP	-10^{-5}	$3.57*10^{-3}$
RL-10	$-2.29*10^{-3}$	$1.52*10^{-3}$
RL-20	$-2.35*10^{-3}$	$1.34*10^{-3}$

Table 4: Total KT-$\overline{DASP}$ growth, averaged over all adaptive sessions of different lengths. Std denotes the standard deviation averaged across all different adaptive session lengths.

We notice in Figure 12 that BP consistently outperforms the RL-10 and RL-20 models across all adaptive session lengths. While BP induces a higher growth across all adaptive sessions, its standard deviation seems to be higher. The standard deviations of the plot indicate that BP can obtain positive growth, while the RL models did not easily manage to obtain positive growth. Looking at Table 4 we confirm that across all adaptive sessions, BP outperforms RL-10 and RL-20. Its standard deviation is wider, but generally, the mean of BP averaged across all adaptive session lengths is larger than RL-10 and RL-20.

RL-10 manages to induce a higher growth than RL-20 across all episodes with different adaptive steps. The exceptions are adaptive step lengths of 1 and 2, where RL-20 outperforms RL-10. While RL-10 typically outperforms RL-20, the difference between the induced growth in models very small. This is also reflected by looking at Table 4, where the total standard deviation of the RL models is very similar, with RL-20 having slightly smaller total standard deviation. The mean seems to be slightly higher for RL-10, meaning that it slightly outperforms RL-20 for inducing overall growth in pupils.

In an adaptive session length of 1 and 2, RL-20 on average induced a higher growth to pupils compared to RL-10. This indicates that RL-20 tends to choose immediate rewards more than RL-10. Specifically, RL-20 chooses immediate rewards on the first two steps, while RL-10 might delay the collected rewards across an episode. It might be that RL-20 chooses more immediate rewards because it is harder to approximate delayed rewards, especially as model horizons get longer. These findings might also indicate that the first two selected exercises are the most important for the pupil in order to pave the way for high growth in subsequent steps. The bigger the model horizon, the more important the first two served exercises seem to be for subsequent pupil growth. It is likely that the first two steps might contain critical states a lot of the time. Choosing appropriate exercises early on in an episode is then crucial for paving a higher growth for pupils.

Looking at Table 4 and Figure 12, we observe that there seems to be a high difference in standard deviations when comparing between the RL models and the BP. One explanation for this is that BP suggested exercises to real pupils; This brings extra third variables to the pupils' knowledge, like, mood, forgetfulness, or focus, which could then account for the larger standard deviation of growth across adaptive steps. This is exemplified in the large standard deviation in big adaptive session lengths, like steps 37 to 40, where the student might have been tired from solving 40 exercises in a row. In contrast, in the simulator, the pupil's answer was simulated by the BP. The BP did not "simulate" the randomness that comes from third variables like forgetfulness, hence providing on average less variance in the accumulated growth of the pupil, with lower standard deviations across adaptive steps. The fact that the BP does not account for the noise induced by the aforementioned third variables is reflected in the almost identical standard deviations across all adaptive session lengths for RL-10 and RL-20. We can also observe the similarity across standard deviations by comparing the total standard deviation across RL-10 and RL-20 models in Table 4. BP originally was made to suggest exercises to pupils. In this sense, it would be more appropriate to simulate the pupil with a knowledge tracing model like Bayesian Knowledge Tracing (BKT), which explicitly learns parameters related to the pupil's cognitive or knowledge state, like "guess", "learn", and "forget" [63]. This could help to simulate the pupil's knowledge better, and therefore create a more realistic simulation environment.

Looking at Figure 12, we notice that as adaptive session length gets bigger, RL-10 and RL-20 models have a smaller standard deviation. We see then that the growth induced by these models stabilizes. It seems that in RL models, the longer an adaptive session gets it becomes more difficult to obtain a higher growth. Contrasting to this is the BP model's induced growth to pupils which does not stagnate as adaptive session lengths get bigger. One explanation for the smaller standard deviation in bigger adaptive session lengths could be a side-effect of simulating the pupil with the BP: as more exercises are served in a session, the state of the pupil tends to get more negative updates. From Equation 27 we see that the state update happens in proportion to how surprised the KT model is with the pupil getting an exercise correctly. In comparison, the BP underestimates the solve probability of the pupil getting the exercise correct. This leads to a lot of proportional negative updates in the KT state of the pupil, which then leads to smaller, usually negative DASP leaps later in an episode. Growth then stagnates, which is reflected in the smaller standard deviations as adaptive sessions get bigger. In comparison then, standard deviations of growth in adaptive sessions of lengths 1 and 2 might be bigger because of the few amount of served exercises. In these adaptive lengths, the KT model does not get "surprised" in each state update as often during an adaptive session, and the DASP leaps tend to have a bigger spread in values.

Note from Figure 12 that no RL model managed to achieve positive growth. Growth converges to values close to zero but is negative. The culprit for this is that in general, negative leaps in DASP growth in the simulator tend to be proportionally higher in magnitude than positive rewards. This leads to typically negative cumulative returns during an episode, and therefore negative $KT\text{-}\overline{DASP}$. This happens because of the aforementioned large and negative proportional updates of the KT state

of the pupil, because of the underestimation of the BP model towards a pupil solving an exercise. This then leads to proportionally larger negative leaps in DASP, as DASP is a function of the KT state and the magnitude of the DASP leaps is directly influenced by the magnitude of the KT state updates. Growth is then kept negative throughout different adaptive session lengths.
While the BP underestimated the pupil's solve probabilities of answering exercises in the simulator, the historical data had real students answering exercises. In the historical data, the KT user states of a pupil were updated according to the answer of a real pupil. This allowed for the accumulation of positive growth, which is observed in Figure 12. We conclude from this that the comparison between simulated pupils and real pupils is unfair. While using the BP to simulate the pupil could be a fairly accurate proxy of the pupil's answers, the underlying underestimation that comes with the model makes gives an advantage over the growth observed in the historical data.

6 Conclusion

In this section, we outline a discussion and the outcomes of the research questions posed in section 1.3. In the first section, the research questions are answered. Then, we provide a general discussion related to our findings and how it relates to the instructional sequencing and reinforcement learning literature. After that, some possible directions for future work are outlined. Lastly, we conclude the thesis.

6.1 Conclusion on Research Questions

When inducing adaptive instructions towards a pupil, should horizon or myopic approaches be preferred for maximizing learning? Based on the investigations of this thesis, we concluded that myopic approaches are preferable for inducing adaptive policies for a pupil. The results of task 2 in Section 5.2 indicate that when comparing pupil growth across three different models, the myopic model consistently induced a higher growth in pupils when compared to horizon-based approaches. Across the RL models with horizons 10 and 20, the first two served exercises were deemed to be the most important in terms of how the pupil will grow over time.

How far should a horizon-based approach look into the future for inducing instructional policies to the pupil? Based on the experiments conducted in the scope of this thesis, models built for inducing instructional policies to a pupil should ideally be myopic. If however there is a need to use RL models that look ahead in a horizon, our results indicate that it might be more beneficial to keep the horizon short. Specifically, our results suggest that a model of horizon 10 slightly outperforms a model of horizon 20. It is worth it to mention that the difference is relatively small and it is possible that there is little to no difference between horizon-based approaches of different look-aheads when considering horizons of 10 and 20.

6.2 Discussion

Our findings support the notion that when constructing a theory of instruction for ITS, the model that aims to solve the underlying MDP should be shorter in its horizon look-ahead, and ideally myopic. This partly resonates with the notion that RL models tend to thrive in more constrained and limited settings for instructional sequencing [5]. Myopic models might be more efficient when constructing ITS in instructional sequencing settings, like the recurrent neural network used in this study.
While this study failed to provide evidence for the effectiveness of RL in instructional sequencing settings, RL models might be more effective in such settings when:

- The action space is small,
- psychological theories are leveraged into the model,
- the model is constrained in some way,
- When the model is evaluated, pupils have little to no prior knowledge of the topic they wish to learn.

Studies in which RL models beat the baseline models in instructional sequencing settings typically leveraged one or more of the above features [5]. As we discuss in detail in the upcoming paragraphs, our RL model or environment structure did not consider any of these features.

Our results resonate with the observation in the literature that models that perform better than baselines tend to leverage psychological theories, and as such, are model-based [5]. Our RL model was purely data-driven, while the BP baseline leveraged psychological theories, namely handing out exercises in the zone of proximal development. Therefore, this study can also be thought of as possible evidence for the effectiveness of leveraging psychological theories in instructional sequencing. It might be likely then, that leveraging psychological theories in RL models will make them more eligible for longer horizons, as model restriction might reduce compounding errors that occur from the horizon lookaheads. The way forward seems then to find a golden standard between two ends, one being mainly leveraging psychological principles of learning (e.g ACT-R) and the other being data-driven approaches, such as model-free RL for instructional sequencing [64]. Combining the modeling constraints of psychological theories, with the powerful potential of big data and individualizing sequencing for pupils according to their needs, holds the promise of powerful intelligent tutoring systems in the future.

RL in instructional sequencing settings might not be suitable when the action space is significantly big, even if we have attempted to scale the model in working on big action spaces by leveraging information about the difficulty of the exercises and using the Wolpertinger architecture. RL models in instructional sequencing tend to perform well with a significantly small action space, e.g up to 50 actions [5]. A big action space typically leads to RL models underperforming because they are data-hungry, and a large action space would demand a huge amount of data for the policy to sufficiently explore all possible state-action pair combinations. As such, our dataset contained 53M interactions, which might not have been enough in terms of covering all these state-action pairs, and subsequently discovering an optimal policy.

It is also worth questioning whether RL model-free methods are effective when the student has little prior knowledge over the subject matter. The data-driven exploration properties of RL could help them explore and bring the pupil's knowledge up to a sufficient level. According to a meta-analysis conducted by Doroudi, RL models for instructional sequencing that tend to perform better than the baseline were generally deployed online to pupils that had little to no prior knowledge of the subject matter [5]. In contrast, we evaluated a given pupil's induced growth from the RL model by initializing episodes at a point where pupils were sufficiently knowledgeable about the subject matter. As an obvious limitation, mastering an already close-to-mastered subject was hard, which partially explains the negative growth shown by the RL models.

Various authors have pointed out an aptitude-treatment interaction effect in favor of low-performing students in RL settings; when students were working through certain educational content in an ITS, with an RL model designing the instructional policy, students that had more room to improve benefited more from a better RL induced instructional policy, compared to already competent students [5, 65, 66]. This could be an indication that RL methods are more effective for students that have little knowledge or are less skilled compared to their more competent counterparts. While in the scope of this study, RL models were trained offline, RL models when deployed and trained online tend to enhance learning in pupils that are underperforming or have little to no prior knowledge of the subject. Training policies offline and evaluating them on the simulator can be seen as the first step of training policies, before actually deploying them and training them further online. Doing this can benefit the learning growth of low-performing pupils as we mitigate the risk of deploying an underperforming policy, as well as then taking advantage of the observed aptitude-treatment interaction effect.

One of the findings of this study was that the first suggested exercises are the most important in order to pave learning growth for pupils. This can be evidence that the structure of the environment contains critical states. Critical states, as first mentioned in 5.1, are states in which the eligible actions are few, and taking a wrong action can lead to regions with negative rewards. It should also be mentioned in

offline scenarios, using offline RL is less preferred over behavioral cloning, for environment structures with a large number of critical states [62].
One likely cause of critical states in instructional sequencing is that rewards are usually delayed due to the complex nature of learning, and therefore finding regions with positive rewards can be difficult [67]. A reason that myopic approaches are effective in instructional sequencing could be because they do not plan ahead, but aim to find the best next action with the highest reward, therefore potentially ”escaping” critical states more easily compared to horizon-based approaches.
While it seems that critical states are prevalent in our formulated environment, generalizations cannot be easily drawn about whether critical states exist in all instructional sequencing settings, as constructed environments in ITS differ from one to another. To overcome critical states in the environment structure, it is important to give the pupil some degree of control over choosing the next exercise. Firstly, learner control gives the pupil motivation in interacting with the ITS, which can significantly boost learning results [68]. One beneficial approach could be for a pupil to choose the next exercise when we have identified the current state as a critical state. This can improve exploration for RL models, as well as potentially escape critical states, specifically by the benefits of control of the pupil over the content that the ITS provides [68]. RL approaches exist in the literature that, for example, aim to find critical states by identifying the variance of the Q-function across all current eligible actions given the current state [69]. Mitigating critical states in such a way could be a way to make RL approaches more effective when applying them in instructional sequencing settings.
One of many significant design considerations for constructing RL models is the reward function. The reward function plays a significant role in an RL model's effectiveness, where slight modifications can result in a significant drop in performance [70]. Reward functions in instructional sequencing, focus but are not limited to capturing the learning gains of the pupil, by realizing them as the solve probabilities before and after answering an exercise, much like we implemented in this study [3]. One way of leveraging psychological theories into an RL model would be by shaping an appropriate reward function; For example, shaping the reward function according to the principle of spaced repetition (exercises not seen by the pupil for a longer time tend to have a bigger associated reward) could improve RL models in instructional sequencing [30].
While it is a good idea to shape the reward function to leverage psychological theories, the reward function should also be shaped according to the learning task at hand. For example, making a reward function that encourages spaced repetition can be beneficial for memorization tasks, where the pupil aims to simply memorize information, like language vocabulary [71]. On the other hand, spaced repetition might not be as beneficial in tasks that require some semantic manipulation of information, for example mathematics [72]. It is then a good idea to shape the reward function according to the learning task at hand.

6.3 Future Work and Limitations

While we trained and evaluated our RL models purely offline, this came with a significant drawback; distributional shift is a problem that has not been overcome, and significantly affects the efficiency of offline RL algorithms. Therefore, we cannot really know the RL models' performance until they are actually evaluated online. Therefore, the direction for future work should be to compare pupils in real life. This would mean that the RL models are trained and evaluated online, and distributional shift, as well as the problems that come with simulating a pupil in the simulator, should be overcome.
Another problem of this study was that when initializing an episode in the simulator, the knowledge of the pupil was already up to par. This then made it difficult for the RL models to achieve positive ”leaps” in learning gains for the pupil. Researchers that want to replicate this study could potentially

deploy and train RL models online, to students with no prior knowledge of the subject matter which they wish to learn. They could then test whether RL models are effective in instructional sequencing settings exactly because of their exploratory properties, especially in the beginning of training RL algorithms where usually exploration takes more emphasis than exploitation.

It is also worth mentioning that comparing simulated pupil growth with growth observed in the historical data is not a fair comparison. As discussed in 5.2, one significant drawback of simulating pupil growth, was that the simulated pupil's solve probability of solving exercises was underestimated. This made the underlying RL policies look like they performed worse, but it could be possible that they perform better than the baseline when they are deployed online. Future work should use state-of-the-art knowledge tracing models when simulating a pupil in the simulator. Specifically, we discussed that in this thesis, simulating the student's answers with BP might not have been optimal, leading to a lack of pupil growth when evaluating the pupil's growth in the simulator. it is important to use a model that approximates parameters related to the pupil's cognitive state, like the ”guess”, ”know”, and ”forget” parameters that BKT and HOT-DINA contain [63, 73]. The result will then be a simulator that has a more accurate approximation of reality.

While we used state-of-the-art models in Knowledge tracing to represent the pupils' state, we did not leverage any psychological theories for the RL models, which are suggested to be important for making RL work in the educational domain [5]. Future direction should therefore enrichen the state representation of the pupil by leveraging psychological theories. For example, the state representation could be enriched by keeping track of time when visiting a certain skill, and enforcing spaced repetition, a well-documented and empirically proven learning approach [30]. By leveraging psychological theories, it might be very likely that horizon-based RL models will perform better on longer horizons, and possibly outperform myopic baselines. The leverage of psychological theories could provide model restrictions that reduce compounding approximation errors that occur during longer horizon lookaheads in RL models.

Researchers that want to replicate this study could also test policies with more horizons. While this study provided evidence that models with myopic horizons are better for instructional sequencing settings, it would be interesting to test whether a policy with a horizon of 15 or models with horizons longer than 20 will induce good policies in pupils. Kumar claims that offline RL algorithms are favorable to BC in environments with a longer horizon, when the data are noisy-expert, meaning data that are expert but sometimes venture away from the expert distribution [62]. In this case, researchers that are testing longer horizons should also test RL algorithms with bigger α which will force the RL algorithms to behave more like offline RL than BC [42].

To the best of our knowledge, no study before has trained RL models with an action space of 100k exercises. The second study with the biggest action space recorded in the literature contained 16k exercises [7]. While we believe that we tackled the problems of a huge action space efficiently by 1) implementing the Wolpertinger architecture and 2) leveraging information about the difficulty of the exercises themselves, future studies should investigate more ways of leveraging psychological theories into RL models so that they can sequence instructions when the action space is significantly big. Restricting models in such a way holds the potential of scaling RL models into huge action spaces.

All previous studies that combined instructional sequencing with RL involved training models either in a simulator or online [3, 7, 5, 14]. The promise that offline RL brings in exploiting large amounts of data in order to construct an optimal policy should be explored further. In the field of offline RL and robotics, attempts are being developed to create practical workflows for offline RL training, RL model selection, and environment construction all the way to RL model deployment. This is analogous to the relatively well-understood workflows for supervised learning problems [74, 75]. Such attempts should be carried over in the domain of offline RL and instructional sequencing, where workflows for environment configuration, model selection, and policy evaluation should be developed. There is a need to discover principles of how to choose offline RL models for deployment in ITS platforms. This will provide efforts to researchers on how to navigate the problem of creating effective RL models for instructional sequencing and ITS.

6.4 Conclusion

In this thesis, we investigated horizon-based vs myopic approaches for instructional sequencing in offline reinforcement learning. Our purpose was to aid researchers in constructing MDPs for instructional sequencing by providing evidence for choosing between horizon-based or myopic models during model selection.

Myopic approaches tend to aid pupil growth more compared to horizon-based approaches. In this sense, while a myopic approach might work best, when choosing horizon-based approaches, there is evidence that shorter horizons work better compared to longer horizons. Our results suggest that the first two served exercises are important for facilitating a pupil's learning growth.

Bibliography

[1] A. E. Blandford, “Intelligent tutoring systems: Lessons learned: Edited by joseph psotka, l. dan massey and sharon a. mutter. lawrence erlbaum, hillsdale, n.j. 1988. 552 pp,” *Computers in Education*, vol. 14, pp. 544–545, 1990.

[2] R. Nkambou, J. Bourdeau, and V. Psyché, *Building Intelligent Tutoring Systems: An Overview*, vol. 308, pp. 361–375. 09 2010.

[3] J. Subramanian, “Deep reinforcement learning to simulate, train, and evaluate instructional sequencing policies,” 2021.

[4] Y. Pu, C. Wang, and W. Wu, “A deep reinforcement learning framework for instructional sequencing,” in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 5201–5208, 2020.

[5] S. Doroudi, V. Aleven, and E. Brunskill, “Where’s the reward?: A review of reinforcement learning for instructional sequencing,” *International Journal of Artificial Intelligence in Education*, vol. 29, 11 2019.

[6] A. T. Corbett and K. Koedinger, “Chapter 37 – intelligent tutoring systems,” 1997.

[7] J. Whitehill and J. Movellan, “Approximately optimal teaching of approximately optimal learners,” *IEEE Transactions on Learning Technologies*, vol. 11, no. 2, pp. 152–164, 2018.

[8] F. Ritter, J. Nerb, E. Lehtinen, and T. O’Shea, *In Order to Learn: How the sequence of topics influences learning*. 08 2007.

[9] P. A. Cohen, J. A. Kulik, and C.-L. C. Kulik, “Educational outcomes of tutoring: A meta-analysis of findings,” *American Educational Research Journal*, vol. 19, no. 2, pp. 237–248, 1982.

[10] C. Reigeluth, *What is Instructional Design Theory and How Is it Changing? (93)*, vol. 2, pp. 5–29. 01 1999.

[11] R. C. Atkinson, “Optimizing the learning of a second-language vocabulary.,” *Journal of Experimental Psychology*, vol. 96, pp. 124–129, 1972.

[12] R. Howard, *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology, 1960.

[13] R. Smallwood, *A Decision Structure for Teaching Machines*. M.I.T. Press research monographs, M.I.T. Press, 1962.

[14] A. Segal, Y. David, J. Williams, K. Gal, and Y. Shalom, “Combining difficulty ranking with multi-armed bandits to sequence educational content,” 04 2018.

[15] M. V. Yudelson, K. R. Koedinger, and G. J. Gordon, “Individualized bayesian knowledge tracing models,” in *Artificial Intelligence in Education* (H. C. Lane, K. Yacef, J. Mostow, and P. Pavlik, eds.), (Berlin, Heidelberg), pp. 171–180, Springer Berlin Heidelberg, 2013.

[16] J.-J. Vie and H. Kashima, “Knowledge tracing machines: Factorization machines for knowledge tracing,” 2018.

[17] S. Sani, T. Mohd Aris, and M. Sulaiman, “Student modeling: An overview,” *International Journal of Advances in Computer Science Its Applications*, vol. 5, pp. 2250–3765, 10 2015.

[18] Y. Bergner, S. Oschler, G. Kortemeyer, S. Rayyan, D. Seaton, and D. Pritchard, “Model-based collaborative filtering analysis of student response data: Machine-learning item response theory,” *5th International Conference on Educational Data Mining*, 01 2012.

[19] G. Abdelrahman, Q. Wang, and B. P. Nunes, “Knowledge tracing: A survey,” 2022.

[20] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[21] J.-J. Vie and H. Kashima, “Knowledge tracing machines: Factorization machines for knowledge tracing,” 2018.

[22] Y. Pu, C. Wang, and W. Wu, “A deep reinforcement learning framework for instructional sequencing,” in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 5201–5208, 2020.

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

[24] K. Katsikopoulos, D. Fisher, and S. Duffy, “Experimental evaluation of policies for sequencing the presentation of associations,” *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 31, pp. 55–59, 01 2001.

[25] Y. Pu, C. Wang, and W. Wu, “A deep reinforcement learning framework for instructional sequencing,” in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 5201–5208, 2020.

[26] J. A. Self, “Bypassing the intractable problem of student modelling,” 1988.

[27] E. J. Sondik, “The optimal control of partially observable markov processes over the infinite horizon: Discounted costs,” *Operations Research*, vol. 26, no. 2, pp. 282–304, 1978.

[28] D. Wammes, B. Slof, W. Schot, and L. Kester, “Teacher judgement accuracy of technical abilities in primary education,” *International Journal of Technology and Design Education*, 02 2022.

[29] N. Lambert, K. Pister, and R. Calandra, “Investigating compounding prediction errors in learned dynamics models,” 2022.

[30] S. Kang, “Spaced repetition promotes efficient and effective learning: Policy implications for instruction,” *Policy Insights from the Behavioral and Brain Sciences*, vol. 3, 01 2016.

[31] E. Pashenkova, I. Rish, and R. Dechter, “Value iteration and policy iteration algorithms for markov decision problem,” 01 1997.

[32] R. Bellman, “Dynamic programming and stochastic control processes,” *Information and Control*, vol. 1, no. 3, pp. 228–239, 1958.

[33] G. Rummery and M. Niranjan, “On-line q-learning using connectionist systems,” *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” 2013.

[35] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” 2015.

[36] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” 2018.

[37] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, “Revisiting fundamentals of experience replay,” 2020.

[38] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in Neural Information Processing Systems 12*, vol. 12, pp. 1057–1063, MIT Press, 2000.

[39] H. Berenji and D. Vengerov, “A convergent actor-critic-based frl algorithm with application to power management of wireless transmitters,” *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 4, pp. 478–485, 2003.

[40] P. Aumjaud, D. McAuliffe, F. J. Rodrí guez-Lera, and P. Cardiff, “Reinforcement learning experiments and benchmark for solving robotic reaching tasks,” in *Advances in Intelligent Systems and Computing*, pp. 318–331, Springer International Publishing, nov 2020.

[41] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” 2015.

[42] S. Fujimoto and S. S. Gu, “A minimalist approach to offline reinforcement learning,” 2021.

[43] S. Levine, A. Kumar, G. Tucker, and J. Fu, “Offline reinforcement learning: Tutorial, review, and perspectives on open problems,” 2020.

[44] A. Kumar, J. Fu, G. Tucker, and S. Levine, “Stabilizing off-policy q-learning via bootstrapping error reduction,” 2019.

[45] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, “Sim-to-real: Learning agile locomotion for quadruped robots,” 2018.

[46] A. Nair, A. Gupta, M. Dalal, and S. Levine, “Awac: Accelerating online reinforcement learning with offline datasets,” 2020.

[47] S. Levine, A. Kumar, G. Tucker, and J. Fu, “Offline reinforcement learning: Tutorial, review, and perspectives on open problems,” 2020.

[48] F. Torabi, G. Warnell, and P. Stone, “Behavioral cloning from observation,” 2018.

[49] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations,” 2017.

[50] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems: Techniques, Applications, and Challenges*, pp. 1–35. New York, NY: Springer US, 2022.

[51] P. Resnick and H. R. Varian, “Recommender systems,” *Commun. ACM*, vol. 40, pp. 56–58, 1997.

[52] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, “Deep reinforcement learning in large discrete action spaces,” 2015.

[53] A. Kumar, A. Zhou, G. Tucker, and S. Levine, “Conservative q-learning for offline reinforcement learning,” 2020.

[54] Z. Zhao, Y. Liang, and X. Jin, “Handling large-scale action space in deep q network,” in *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pp. 93–96, 2018.

[55] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[56] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, “Learning values across many orders of magnitude,” 2016.

[57] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.

[58] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[59] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[60] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[61] W. McKinney, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51 – 56, 2010.

[62] A. Kumar, J. Hong, A. Singh, and S. Levine, “When should we prefer offline reinforcement learning over behavioral cloning?,” 2022.

[63] M. V. Yudelson, K. R. Koedinger, and G. J. Gordon, “Individualized bayesian knowledge tracing models,” in *Artificial Intelligence in Education* (H. C. Lane, K. Yacef, J. Mostow, and P. Pavlik, eds.), (Berlin, Heidelberg), pp. 171–180, Springer Berlin Heidelberg, 2013.

[64] F. Ritter, F. Tehranchi, and J. Oury, “Act-r: A cognitive architecture for modeling cognition,” *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 10, p. e1488, 12 2018.

[65] V. Aleven, E. A. McLaughlin, A. Glenn, and K. Koedinger, “Instruction based on adaptive learning technologies,” 2016.

[66] S. Shen, M. Ausin, B. Mostafavi, and M. Chi, “Improving learning reducing time: A constrained action-based reinforcement learning approach,” pp. 43–51, 07 2018.

[67] S. Ju, G. Zhou, M. Abdelshiheed, T. Barnes, and M. Chi, “Evaluating critical reinforcement learning framework in the field,” in *International conference on artificial intelligence in education*, pp. 215–227, Springer, 2021.

[68] M. B. Kinzie and H. J. Sullivan, “Continuing motivation, learner control, and cai,” *Educational Technology Research and Development*, vol. 37, no. 2, pp. 5–14, 1989.

[69] I. Karino, Y. Ohmura, and Y. Kuniyoshi, “Identifying critical states by the action-based variance of expected return,” in *Artificial Neural Networks and Machine Learning – ICANN 2020*, pp. 366–378, Springer International Publishing, 2020.

[70] M. Farsang and L. Szegletes, “Importance of environment design in reinforcement learning: A study of a robotic environment,” 02 2021.

[71] E. Chukharev-Hudilainen and T. A. Klepikova, “The effectiveness of computer-based spaced repetition in foreign language vocabulary instruction: A double-blind study,” *Calico Journal*, vol. 33, no. 3, pp. 334–354, 2016.

[72] H. Pashler, D. Rohrer, N. J. Cepeda, and S. K. Carpenter, “Enhancing learning and retarding forgetting: Choices and consequences,” *Psychonomic bulletin & review*, vol. 14, no. 2, pp. 187–193, 2007.

[73] Y. Xu and J. Mostow, “A unified 5-dimensional framework for student models,” *CEUR Workshop Proceedings*, vol. 1183, pp. 122–129, 01 2014.

[74] A. Kumar, A. Singh, S. Tian, C. Finn, and S. Levine, “A workflow for offline model-free robotic reinforcement learning,” 2021.

[75] H. Wang, A. Sakhadeo, A. White, J. Bell, V. Liu, X. Zhao, P. Liu, T. Kozuno, A. Fyshe, and M. White, “No more pesky hyperparameters: Offline hyperparameter tuning for rl,” 2022.

6.5 DDPG Training Pseudocode

Algorithm 6 Deep Deterministic Policy Gradient (DDPG)

```
 1: Input: initial policy parameters $\theta$, Q-function $\phi$, empty replay buffer D
 2: set target parameters equal to main parameters $\theta_{targ} \rightarrow \theta$, $\phi_{targ} \rightarrow \phi$
 3: **repeat**
 4:     Observe state s and select action $a = clip(\mu_{\theta(s)} + \varepsilon, a_{Low}, a_{High})$, where $\varepsilon \sim N$
 5:     Execute a in environment
 6:     Observe next state s', reward r, and done signal d to indicate whether s' is terminal
 7:     Store (s,a,r,s',d) in replay buffer D
 8:     if s' is terminal, reset environment state
 9:     **if** its time to update **then**
10:         **for** j in range(however many updates) **do**
11:             Randomly sample a batch of transitions, $B = \{(s,a,r,s',d\}$ from D
12:             Compute targets
                    $$y = r + \gamma(1-d)Q_{\phi targ,i}(s', \mu\theta_{targ}(s'))$$
13:             update Q-function by one step of gradient descent:
                    $$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s,a) - y)^2$$
14:             Update policy with gradient ascent:
                    $$\nabla_\theta \frac{1}{|B|} Q_\phi(s, \mu_\theta(s))$$
15:             Update target networks with
                    $$\phi_{targ} \leftarrow \rho\phi_{targ} + (1-\rho)\phi,$$
                    $$\theta_{targ} \leftarrow \rho\theta_{targ} + (1-\rho)\theta,$$
16:
17:         **end for**
18:     **end if**
19: **until** convergence
```