# Pick the Moment: Identifying Critical Pedagogical Decisions Using Long-Short Term Rewards

Song Ju, Guojing Zhou, Tiffany Barnes, Min Chi
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
{sju2, gzhou3, tmbarnes, mchi}@ncsu.edu

## ABSTRACT

Identifying critical decisions is one of the most challenging decision-making problems in real-world applications. In this work, we propose a novel Reinforcement Learning (RL) based Long-Short Term Rewards (LSTR) framework for critical decisions identification. RL is a machine learning area concerning with inducing effective decision-making policies, following which result in the maximum cumulative *reward*. Many RL algorithms find the optimal policy via estimating the optimal Q-values, which specify the maximum cumulative reward the agent can receive. In our LSTR framework, the *long term* rewards are defined as *Q-values* and the *short term* rewards are determined by the *reward function*. Experiments on a synthetic GridWorld game and real-world Intelligent Tutoring System datasets show that the proposed LSTR framework indeed identifies the critical decisions in the sequences. Furthermore, our results show that carrying out the critical decisions alone is as effective as a fully-executed policy.

## Keywords

Critical Decisions, Pedagogical Strategies, Critical Reinforcement Learning, Reinforcement Learning

## 1. INTRODUCTION

People make decisions every day, from minor decisions such as what to eat for lunch, to major decisions such as which college to enroll. This is equally true for tutorial interactions. Some decisions, such as what type of example to use may be minor, while others such as whether to give a new problem, or provide a solution for an old one, may not. In many cases the true significance of these decisions will not be known until well after the fact (much delayed), when students' exam scores come in or beyond. Moreover, for many such decisions, the significance is often individualized. So our research question is: ***Given a long trajectory of decisions, can we automatically identify those which are critical to the outcome?***

Our work is primary concerned with identifying critical decisions in interactive learning environments such as Intelligent Tutoring Systems (ITSs) and educational games, where the human-agent interactions can be viewed as a temporal sequence of steps [2, 14]. Most ITSs are *tutor-driven* in that *the tutor* decides what to do next. For example, the tutor can *elicit* the subsequent step from the student either with prompting or without (e.g., in a free form entry window where each equation is a step). When a student enters an entry on a step, the ITS records its success or failure and may give feedback (e.g. correct/incorrect markings) and/or hints (suggestions for what to do next). Alternatively, the tutor can choose to *tell* them the next step directly. Each of such decisions affects the student's successive actions and performance and some may be more impactful than others. *Pedagogical policies* are used for the agent (tutor) to decide what action to take next in the face of alternatives.

Reinforcement Learning (RL) offers one of the most promising approaches to data-driven decision-making for improving student learning in ITSs. RL algorithms are designed to induce effective policies that determine the best action for an agent to take in any given situation so as to maximize a cumulative reward. In recent years, RL, especially Deep RL, has achieved superhuman performance in several complex games [25, 26, 3]. However, different from the classic gameplay situations where the ultimate goal is to make the agent effective, in human-centric tasks such as ITSs, the ultimate goal is for the agent to make the *student-system interactions* productive and fruitful. A number of researchers have studied the application of existing RL algorithms to improve the effectiveness of ITSs [5, 24, 16, 21, 20, 19, 6, 27, 10, 31, 30, 32]. While promising, relatively little work has been done to analyze, interpret, explain, or generalize RL-induced policies. While traditional hypothesis-driven, cause-and-effect approaches offer clear conceptual and causal insights that can be evaluated and interpreted, RL-induced policies are often large, cumbersome, and difficult to understand. The space of possible policies is exponential in the number of domain features. It is therefore difficult to identify the system decisions that critical to desirable outcomes. This raises a major open question: *How can we identify the critical system interactive decisions that are linked to student learning?*

In this work, we propose Long-Short Term Rewards (LSTR) framework to identify *critical* decisions based on RL-induced policy. For RL-induced policies, we explore Deep Q-Networks (DQNs) [18] and also modify Deep Q-Networks based on

critical decisions referred as Critical DQN in the following. More specifically, we define critical decisions as those optimal decisions have to be made for the desired outcomes. To quantify their impacts, we define critical policy as the one which will carry out the optimal actions on the critical decisions while randomly on others. To identify critical decisions, we investigate on using an RL-induced policy's action-value functions (long term) alone and using both action-value functions (long term) and immediate rewards (short-term). The effectiveness of the proposed LSTR framework is evaluated on a synthetic GridWorld game and real-world Intelligent Tutoring System datasets. Our results show that the proposed LSTR framework indeed identifies critical decisions and moreover, carrying out the critical decisions alone is as effective as a fully-executed policy.

Our main contributions are summarized as follows: 1) we proposed the Long Short Term Rewards framework to identify critical decisions and evaluated on both a synthetic GridWorld game and real-world ITS dataset. 2) we proposed Critical DQN to improve the long term rewards in identifying critical decisions and investigated its advantages and disadvantages.

# 2. METHOD

We follow the conventional Reinforcement Learning (RL) notation. An agent interacts with an environment over a series of decision-making steps. The environment is framed as a Markov Decision Process (MDP). At each timestep $t$, the agent observes the state the environment is in, denoted $s_t$; then the agent chooses an action from a discrete set of possible actions: $A \in (a_1, a_2, ..., a_n)$. As a result, the environment provides a scalar *immediate reward* $r$. We assume that the future rewards are discounted by the factor $\gamma \in (0, 1]$, and the agent's goal is to maximize the expected discounted sum of future rewards, also known as the return. The return at time-step $t$ is defined as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where $T$ is the last time-step in the episode.

The goal of the agent is to find the optimal action-value function $Q^*(s, a)$, which will result in the agent receiving the highest possible expected return, starting from state $s$, taking action $a$, and following the optimal policy $\pi^*$ thereafter. Formally, we define the optimal action-value function as $Q^*(s, a) = \max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$. The optimal action-value function must follow the *Bellman Equation* shown in Equation 1, which states that the Q-value for a given state and action should be equal to the immediate reward obtained after taking that action, plus the discounted Q-value of the optimal action $a'$ taken from the next state $s'$. Note that this is an expectation over the next states sampled from the environment.

$$Q^*(s, a) = \mathop{\mathbb{E}}_{s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q^*(s', a') | s, a] \tag{1}$$

In our case, we follow the batch Reinforcement Learning formulation in that we have a fixed-size dataset $\mathcal{D}$ consisting of all historical sample episodes and each episode is denoted as $s_1 \xrightarrow{a_1, r_1} s_2 \xrightarrow{a_2, r_2} s_3 \xrightarrow{a_3, r_3} \cdots s_L$). To make this task more general, we assume that the state distribution and behavior policy that were used to collect this data are both unknown.

In the following, we will describe the two DRL algorithms explored: DQN and Critical DQN and two ways of defining critical decisions: long term reward vs. long-short term reward. Based on two types of DRL methods and two ways of identifying critical decisions, we will compare six different policies.

## 2.1 Two Types of Deep RL Policy

### 2.1.1 Original DQN

Deep Q-Network (DQN) is one of most promising approaches which is widely used on areas like robotics and video games [18]. Fundamentally, DQN is a version of Q-learning which uses neural networks to approximate the Q-values of the different state-action couples. In order to train the DQN algorithm, the two neural networks with equal architectures are employed: one for calculating the Q-value of the current state and action: $Q(s, a)$ and another neural network to calculate the Q-value of the next state and action: $Q(s', a')$. The former is the main network and its weights are denoted by $\boldsymbol{\theta}$ and the latter is the target network, and its weights are denoted by $\boldsymbol{\theta}^-$. The *Bellman Equation* for DQN is shown in Equation 2 and it is trained through running a gradient descent algorithm to minimize the squared difference of the two sides of the equality.

$$Q(s, a; \boldsymbol{\theta}) = \mathop{\mathbb{E}}_{s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^-)] \tag{2}$$

The main network is trained on every training iteration, while the target network is frozen for a number of training iterations. Every $k$ training iterations, the weights of the main neural network are copied into the target network. This is one of the techniques used in order to avoid divergence during the training process. In practice, DQN also uses an experience replay buffer to store the recently collected data and to uniformly sample $(s, a, r, s')$ steps from it. By sampling uniformly, it breaks the correlations between samples of the same episode, making the learning process more robust and stable. In this work, as we are doing batch RL, our whole dataset will be the experience replay buffer, and it will not change during the training process.

Basically, DQN is a Q-leaning method that it finds the optimal action-value function by updating its action-value function approximator recursively. Its major difference from the traditional RL is that a deep neural network is used as action-value function approximator and this allows it to deal with the tasks with high dimensional state space.

### 2.1.2 Critical DQN

In the original DQN, the Q functions are estimated based on the assumption that the optimal policy will be followed to the end. We define *critical policy to be the one that the optimal decision will be carried out on critical decision points while random decisions on the rest.* By not taking the optimal actions on non-critical decisions, we fundamentally change the dynamics of Bellman equation which assumed full-execution of the policy. Therefore we need to modify it so that it can incorporate our critical decisions into consideration.

For a single $(s, a, r, s')$ tuple, the original Bellman Equation can be expressed as:

$$Q(s,a) = r + \gamma \max_{a'} Q(s', a') \tag{3}$$

where $r$ is the immediate reward for taking action $a$ at state $s$; $\gamma$ is the discount factor; and $Q(s', a')$ is the action-value function for taking action $a'$ at the subsequent state $s'$.

To induce a critical policy, we will modify the original Bellman equation based on whether a decision is critical or not. The intuition behind the Critical DQN is that if a decision is important, then the agent should take the best action otherwise the agent can randomly choose an action to take. Therefore, we have:

$$Q(s,a) = \begin{cases} r + \gamma maxQ(s', a') & \text{s' is critical} \\ r + \gamma meanQ(s', a') & \text{s' is non-critical} \end{cases} \tag{4}$$

In equation 4, to update the Q-value for any given $s$ and $a$, it will consider whether the next state $s'$ is critical or not. If it is critical, the maximum Q-value for $s'$ will be used to update Q(s,a); if the decision $s'$ is non-critical, then the average Q-value among all the actions on $s'$ will be used to update Q(s,a).

Algorithm 1 presents the pseudo-code for critical DQN. First, it initializes all Q-values using the immediate rewards to avoid the bias of the neural network. In the main training loop, for each iteration, the algorithm first calculate the median threshold of Q-value difference over all the states. Then, for each $(s, a, r, s')$ tuple, if the Q-value difference of $s'$ is larger than the median threshold, we consider the decision on that state is critical and its value function is $max_{a'} Q(s', a'; \theta^-)$; for non-critical decisions, their value function are defined as $mean_{a'} Q(s', a'; \theta^-)$. In this work, we assumes that half of the decisions in the training dataset are critical so that the median threshold is applied to separate critical and non-critical decisions quantitatively.

## 2.2 Two Types of Critical Rewards

### 2.2.1 Long Term Rewards (LongTRs)

In RL, $Q(s, a)$ is an estimation of the cumulative future rewards the agent will receive by taking action a at state s and following the policy to the end. If the Q-values for all the actions are the same, then it doesn't matter which action to take because all the actions will result in the same final reward. If the Q-value for one action is much larger than the others, then taking that action will have great impact on the future reward and this decision should be critical. So, the Long Term Reward is defined as how much cumulative future rewards the best action will obtain compared with the worst action. For this paper, we therefore define the Long Term Reward (LongTR) as:

$$LongTR(s) = \max_a Q(s,a) - \min_a Q(s,a) \tag{5}$$

which is the difference between the maximum and minimum Q-values in the state s. In general, the higher the LongTR, the more important the decision is.

**Algorithm 1** Pseudocode of Critical DQN

Initialize the training dataset $D$ as $(s, a, r, s')$ tuples.
Initialize the Q function with random parameters $\theta$
Initialize the target $\hat{Q}$ function with parameters $\theta^- = \theta$

// Initialize $Q(s, a)$ as immediate reward
**for** each $(s_i, a_i, r_i, s'_i)$ in $D$ **do**
    set $y_i = r_i$
**end for**
Perform gradient descent on $(y_i - Q(s_i, a_i; \theta))^2$
Reset $\hat{Q} = Q$

// Main Training Loop
**for** iteration k = 1, 2, ... till convergence **do**
    Initialize empty array $Q_{diffs}$
    **for** each $(s_i, a_i, r_i, s'_i)$ in $D$ **do**
        $Q_{diffs} \leftarrow (maxQ(s_i, a'; \theta^-) - minQ(s_i, a'; \theta^-))$
    **end for**
    $median\_threshold = median(Q_{diffs})$
    **for** each $(s_i, a_i, r_i, s'_i)$ in $D$ **do**
        **if** terminal $s'_i$ **then**
            Set $y_i = r_i$
        **else**
            $Q_{diff} = maxQ(s'_i, a'; \theta^-) - minQ(s'_i, a'; \theta^-)$
            **if** $Q_{diff} > median\_threshold$ **then**
                Set $y_i = r_i + \gamma max_{a'} Q(s', a'; \theta^-)$
            **else**
                Set $y_i = r_i + \gamma mean_{a'} Q(s', a'; \theta^-)$
            **end if**
        **end if**
    **end for**
    Perform gradient descent on $(y_i - Q(s_i, a_i; \theta))^2$
    Every C steps reset $\hat{Q} = Q$
**end for**

### 2.2.2 Long-Short Term Rewards (LSTRs)

For the LongTR, it only considers the cumulative future rewards but not immediate rewards. In a deterministic environment, LongTR is enough to identify critical decisions. But in a stochastic environment like the real world, some non-critical decision points would become critical and the LongTR can't detect their importance. For example, Figure 1 shows a simple MDP with seven states and one reward in the central state. Based on the LongTR, the decisions on S2 and S3 are critical because if doesn't move to the center, the agent will miss the +10 rewards. S1 is not critical based on LongTR because either move up or down doesn't affect collecting the reward as long as the agent takes the right action on state S2 or S3. However, in a stochastic environment, the agent should get the reward as soon as possible because the longer the path, the higher the risk to deorbit the rail. In RL, the LongTR can't learn the importance of state S1 but the immediate reward can. So, immediate rewards are served as Short Term Rewards (ShortTRs) in LSTR to complement the weakness of LongTRs that the agent should collect the rewards immediately without wandering.

## 2.3 Identifying & Evaluating Critical Decision

The effectiveness of our LSTR framework on identifying critical decisions is evaluated by the performance of **critical policy**. Unlike normal RL policies whose decisions are car-

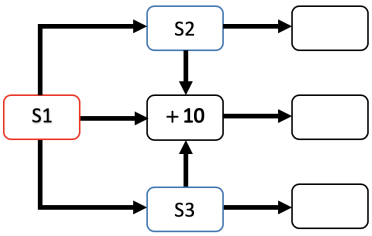


**Figure 1: An MDP Example for LSTR**

ried out all the time, our critical policy only follows the RL policy's decisions at critical points and takes random actions otherwise. Ideally, the more accurate the critical decisions are identified, the better performance the critical policy should have.

More specifically, for a given training data set, we can induce RL policies following the original DQN or the Critical DQN, named as $\pi$ or $\pi_c$, respectively. For each of the policy, there are two ways to identify critical decisions, using LongTR ($L$) and using LSTR ($LS$). Based on the rewards used for critical decision identification and the policy used for execution, we have the following six critical policies shown in the table 1.

**Table 1: Six Critical Policies**

| | Critical Policy | Execution Policy | Rewards for Identifying Critical Decision |
|---|---|---|---|
| 1 | $\pi(L)$ | $\pi$ | LongTRs in $\pi$ |
| 2 | $\pi(LS)$ | $\pi$ | LSTRs in $\pi$ |
| 3 | $\pi_c(L_c)$ | $\pi_c$ | LongTRs in $\pi_c$ |
| 4 | $\pi_c(LS_c)$ | $\pi_c$ | LSTRs in $\pi_c$ |
| 5 | $\pi(L_c)$ | $\pi$ | LongTRs in $\pi_c$ |
| 6 | $\pi(LS_c)$ | $\pi$ | LongTRs in $\pi_c$ |

The first four critical policies are a simple 2 ($\pi$ vs. $\pi_c$) by 2 ($L$ vs. $LS$) combination that each policy uses its own rewards to identify critical decisions. However, the connection between the critical decisions and the performance of critical policy is based on the assumption that the policy carried out at the critical points are optimal. In the Critical DQN, average Q-value is considered in the updating process and this may slow down the convergence. As a result, the policy $\pi_c$ can be non-optimal. So, we include other two critical policies: $\pi(L_c)$ and $\pi(LS_c)$ which using the LongTR and LSTR from $\pi_c$ to identify critical decisions but executing the policy $\pi$ to make decisions. In general, the original DQN should converge faster and generate better policy than the Critical DQN.

# 3. SIMULATION ENVIRONMENT

## 3.1 GridWorld Description

The GridWorld environment is like a maze that the agent learns an optimal path from the start point to the end point. Figure 2 shows our GridWorld environment, which consists of 7 by 14 cells. The agent starts from the start state (right bottom corner), explores the 2D space and finishes at the end state (left upper corner). There are several walls in the GridWorld which are marked as black blocks. The agent state is simply represented by the X and Y coordinates.

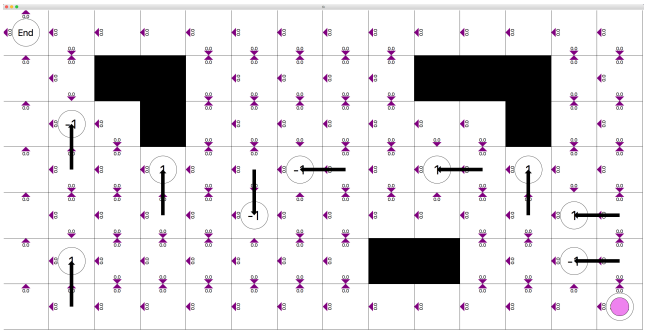


**Figure 2: The Interface of the GridWorld Game**

**Action** At each step, the agent can take three actions: up, down and left. In Figure 2, the possible actions for each state are labeled as small purple triangles that some states have three possible actions while some have two or one possible actions. The possible actions for each state are predefined in the environment so that the agent never hit the wall or the boundary.

**Reward** When moving in the GridWorld, there is -0.1 reward penalty for each step and the agent can collect -1 and +1 rewards. In order to simulate the real world, the reward function is designed in state-action-state way, $R(s, a, s')$. The black arrows indicate that only enter the reward state along with the arrow, the agent can get the reward -1 or +1. Otherwise, the agent won't receive rewards. Furthermore, when the agent hits the reward state, it is forced to move left. This design aims to avoid the agent from collecting the same +1 reward repeatedly without forwarding to the terminal state.

**Deterministic vs. Stochastic** There are two transition settings in the GridWorld game: deterministic and stochastic. For deterministic transition setting, the next state is determined by the current state and action. For stochastic transition setting, the same state-action pair can result in different next states. For example, for deterministic setting, if the agent takes action 'left', then it will move to the left neighbor cell with 100 probability. For the stochastic transition setting, if the agent takes action 'left', then it only has 85% chance moving left, and 15% chance moving to other possible directions.

Finally, the performance of RL-induced policy in the GridWorld is evaluated by the final delayed reward which is the cumulative rewards during a trial. A good RL-induced policy should collect more +1 rewards, avoid -1 rewards and spend less steps to reach the goal.

## 3.2 Experiment Setup

**Data Collection** Since we focus on applying offline RL approaches to induce pedagogical policies, we induce all GridWorld policies offline. Following the data collection procedure in ITS, we collected the training data using a random policy. For the deterministic environment, we collected 500 randomly generated trajectories. Considering that the stochastic environment is more complicated, we collected

1000 trajectories for it.

**Inferring Immediate Rewards** Our LSTR framework requires immediate rewards to identify critical decisions, but our ITS data only have delayed rewards. Thus, we apply a Neural Network (NN) based approach to infer "immediate" rewards from delayed rewards. Given a trajectory to the NN as input, it outputs an "inferred" immediate reward for each step in the trajectory. The NN is trained using an additive error (the mean square error between the sum of inferred immediate rewards and the delayed rewards) as the loss function.

**Critical Threshold Determination** A key thing for identifying critical decisions is to choose an appropriate threshold on the long term and short term rewards that would not include too many trivial decisions but at the same would not exclude too many critical decisions. In order to pick a proper one, we conducted an analysis on the real and inferred immediate rewards and the Q-value difference. For immediate rewards, we would want to collect the large positive rewards and avoid the large negative rewards. Thus, rewards with a large absolute value should be considered as critical. Figure 3 shows the distribution of the real and inferred immediate rewards on the deterministic training dataset. The X axis shows the percentage of decisions in the dataset ranked by the value of the rewards (from large to small), and the Y axis shows value of the rewards. The threshold was set by allowing the real and critical rewards to identify similar numbers of critical decisions, which resulted in the value of 0.5. That is, if the ShortTR of a decision is greater than 0.5 or less than -0.5, it is critical. The same threshold was used for the stochastic dataset.

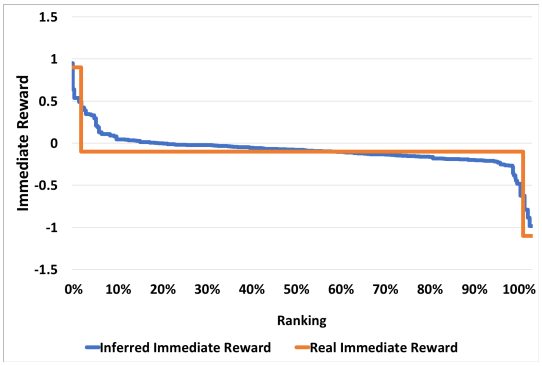


**Figure 3: Immediate Reward Distribution**

With the determined ShortTRs threshold, we explore different thresholds for the LongTRs in the experiment. The larger the threshold is (on percentage ranking), the more decisions will be carried out following the policy and the performance will in turn be better. To find a good balance between the number of critical decisions and the performance of the policy, we apply the policy with different thresholds (on percentage) at a 10% interval from 0% to 100% (0%, 10%, 20% ... 100%). Figure 4 shows an example distribution of the Q-value difference (LongTRs), calculated using the $\pi$ policy in the deterministic dataset. For LSTRs, the critical decisions are the union from two set of critical decisions identified by LongTRs and ShortTRs separately.

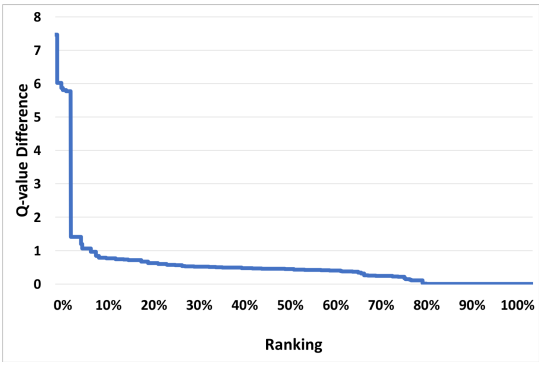


**Figure 4: Q-value Difference Distribution**

## 3.3 Results

We evaluate the performance of the six critical policies across two types of environment (deterministic vs. stochastic) with two types of immediate rewards (real vs. inferred). Figure 5 to Figure 8 shows the online evaluation results for the four possible settings. The X axis shows the percentage of decisions identified as critical ones based on the LongTRs on the training data. For example, 10% means the decisions with the top 10 percent LongTRs were considered as critical. The Y axis shows the cumulative rewards (the average of 100 trials under different random seeds) received by each critical policy. As expected, across all four figures, there is a general trend that the more decisions considered as critical, the better the policy will perform.

Overall, $\pi_c(LS_c)$ and $\pi(LS_c)$ outperforms the other four policies across all four settings. This suggests that when identifying critical decisions, LSTRs are more effective than LongTRs and Critical DQN is more effective than original DQN. This supported our expectation that both long term and short term rewards should be considered in critical decision identification and Critical DQN provides a better estimation of the long term rewards when the policies are partially carried out.

Next, we investigate in detail how the execution policy and the rewards (long term vs. long-short term) may impact the performance of the critical policies. More specifically, we present our results in five parts. First, compare the effectiveness of the original and Critical DQN on LongTRs. Second, investigate whether LSTRs can lead to better performance than LongTRs. Third, a mixed comparison between the critical decision recognition and policy execution. Fourth, exam the effectiveness of the inferred rewards. Finally, explore the performance of the Critical DQN with limited amount of training data.

### *3.3.1 Original DQN vs. Critical DQN on LongTRs*

We first focus on comparing the original DQN policy $\pi(L)$ and the Critical DQN policy $\pi_c(L_c)$ with LongTRs, where the same policy was used for both execution and critical decisions identification. As we can see in all four figures, $\pi_c(L_c)$ outperformed $\pi(L)$ when no more than 50% decisions were considered as critical. More importantly, the fewer the critical decisions, the larger the gap is (except 0% which is totally random). This suggests that the Q-value difference in $\pi_c$ is

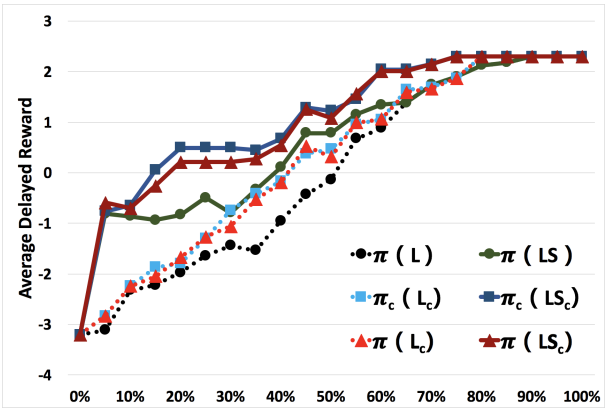


**Figure 5: Deterministic GridWorld with Imm**

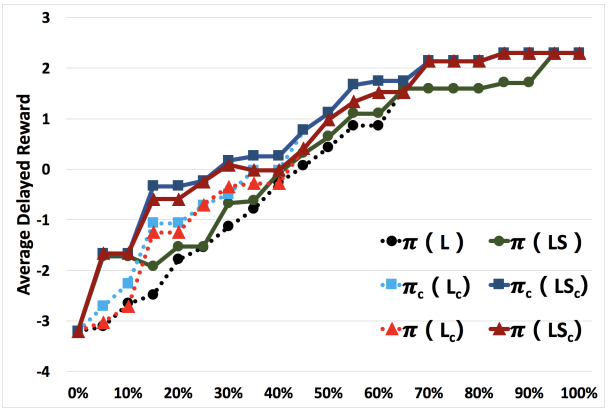


**Figure 6: Deterministic GridWorld with Infer**

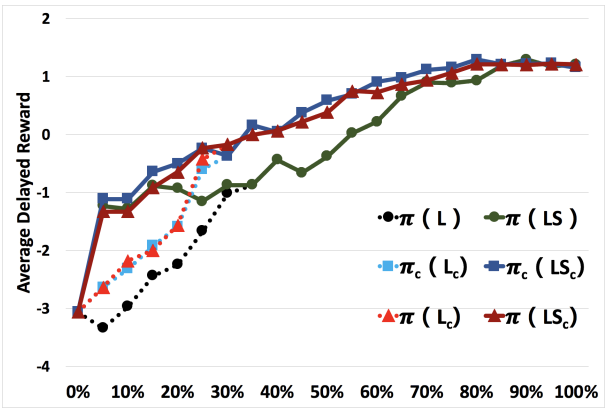


**Figure 7: Stochastic GridWorld with Imm**

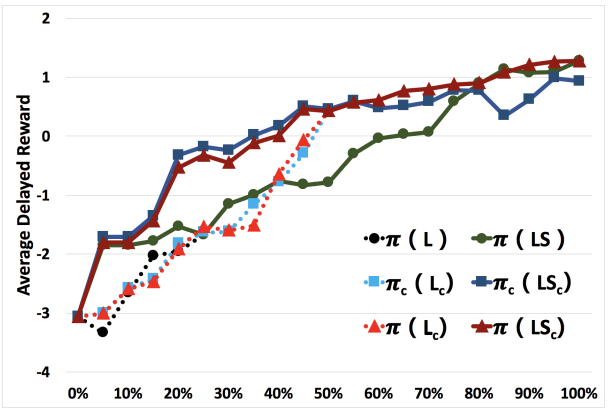


**Figure 8: Stochastic GridWorld with Infer**

more accurate and sensitive than that in $\pi$ in identifying critical decisions. This result is not surprising because the Critical DQN already took the random execution of non-critical decisions into account in policy induction. Additionally, as expected, as the percentage of critical decisions increasing, both policies reached optimal. This suggests that our proposed Critical DQN could generate optimal policies as the DQN can do.

### 3.3.2 *LSTRs vs. LongTRs*

Second, we investigate how different rewards (LSTRs vs. LongTRs) may impact the performance of the policies. LSTR policies are shown in solid lines while LongTR policies are shown in dashed lines across Figure 5 to Figure 8. Here we focus on comparing the pair of policies with the same execution and critical decision identification policies such as $\pi(L)$ vs. $\pi(LS)$ and $\pi(L_c)$ vs. $\pi(LS_c)$. Overall, results showed that the LSTR policies outperformed the LongTR policies when no more than 50% decisions were considered as critical (with few exceptions where the two policies have equal performance). More importantly, the performance of the LSTR policies had a sharp increase in the interval of 0% to 50% while the increase of the LongTR policies was relatively smooth. This resulted in a large gap between them when few decisions were considered as critical. This gap gradually diminished as more decisions were included and disappeared eventually. This suggests that considering both the long term and short term rewards is more effective than considering the long term rewards only, especially when few decisions were considered as critical.

### 3.3.3 *Mixed Comparison*

There are two factors in the critical policy: execution policy and Rewards for critical decision identification. In this comparison, we fixed one factor and examined the impact of the other one on the critical policy. First, through fixing the execution policy as $\pi$, a comparison between $L_c$ vs. $L$ showed that the $\pi(L_c)$ outperformed the $\pi(L)$ across all the four Figures 5 to 8. Similar to this setting, we can get the same results that $\pi(LS_c)$ is better than $\pi(LS)$. It means that the LSTRs in Critical DQN policy is more accurate to identify critical decisions than the original DQN. When fixing the Rewards for critical decision identification as $L_c$, a comparison between $\pi$ vs. $\pi_c$ showed that $\pi(L_c)$ and $\pi_c(L_c)$ have similar performance. This result also applies to $\pi(LS_c)$ and $\pi_c(LS_c)$. It indicates that the policy $\pi$ and $\pi_c$ could make similar decisions on critical points and the Critical DQN could induce optimal policy as the original DQN.

### 3.3.4 *Inferred Rewards vs. Immediate Rewards*

We also examined the effectiveness of the inferred immediate rewards by comparing them with real rewards. Figure 5 and 7 show the policies (immediate critical policy) induced using real immediate rewards; while Figure 6 and 8 show the policies (inferred critical policy) induced using inferred rewards. Through comparing the performance at 100%, all the inferred critical policies could reach the same optimal with the immediate critical policies in both deterministic and stochastic environments. This suggests that the inferred rewards could generate the optimal policy as real rewards. Then, the lines of inferred critical policies in Figure 6 and 8 have similar trend patterns with the lines in Figure 5 and 7, respectively. It means that the LSTRs calculated by inferred critical policies have similar distribution with the ones from immediate critical policies. In sum, the result indicates that inferred rewards could not only generate the optimal policy but also produce reliable LSTRs.

### 3.3.5 *Data-Efficiency for Critical DQN*

From the previous results, we could get a conclusion that when the policies $\pi$ and $\pi_c$ are both optimal, $\pi_c(L_c)$ is better than $\pi(L)$ regarding the identification of critical decisions. But what if we don't have enough data to train an optimal policy, how's the critical DQN performing?

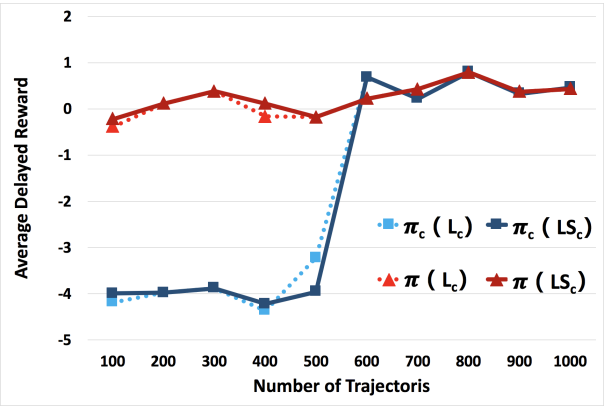


**Figure 9: Original DQN vs. Critical DQN**

Figure 9 shows the online performance of $\pi_c(L_c)$ vs. $\pi(L_c)$ and $\pi_c(LS_c)$ vs. $\pi(LS_c)$ as the number of training trajectories increasing. The X axis is the number of trajectories used to train the critical policies. The Y axis is the cumulative rewards (the average of 100 trails under different random seed) received by each critical policy. In this experiment, we applied the same rule to identify critical decisions for all the four policies and the only difference is which RL policy makes decision on the critical decision points. For $\pi(L_c)$, it means the critical decisions are identified by the LongTRs in $\pi_c$ but execute $\pi$ to make decisions in the online evaluation. It is the same for $\pi(LS_c)$ that the LSTRs come from $\pi_c$ while $\pi$ decides what action to take. More specifically, the threshold for critical decisions is fixed by applying the same 0.5 threshold on short term rewards and 50% threshold on long term rewards.

The result shows that when the training datset is less than 600 trajectories, the Critical DQN policies are worse than the original DQN policies. When the training dataset is larger than 600 trajectories, they have similar performance. This suggests that the Critical DQN needs more data to converge to the optimal policy. But the LSTRs in Critical DQN is always good as the red lines $\pi(L_c)$ and $\pi(LS_c)$ keep staying in the upper area from 100 to 1000 training trajectories. In summary, the Critical DQN could provide the best LSTRs to identify critical decisions but it needs more data to make good decision.

# 4. REAL-WORLD APPLICATION

## 4.1 Pyrenees Tutor Description

Pyrenees tutor is a web-based ITS for probability. It covers 10 major principles of probability, such as the Addition Theorem and Bayes' Rule. Pyrenees tutor provides step-by-step instruction and immediate feedback. Pyrenees tutor can also provide on-demand hits prompting the student with what they should do next. As with other systems, help in Pyrenees tutor is provided via a sequence of increasingly specific hints. The last hint in the sequence, the bottom-out hint, tells the student exactly what to do.

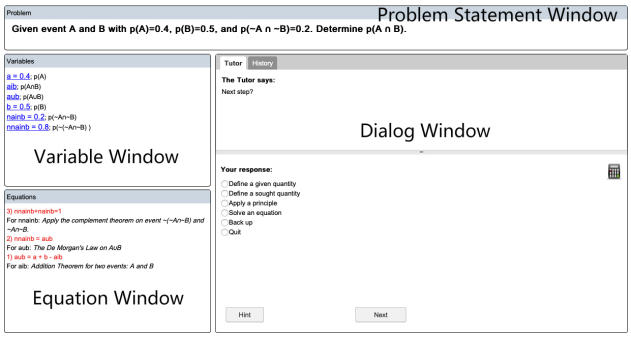


**Figure 10: The Interface of the Pyrenees Tutor**

Figure 10 shows the interface of Pyrenees, which consists of four windows. The top window shows the problem statement and doesn't change throughout the problem. In the dialog window, the upper part shows the instructions the tutor gives to the students such as an explanation of the current step or a prompt for the next step. At the same time, student enters an answer in the lower part of the dialog window such as selecting a choice or writing an equation. Any variables or equations generated through this process are shown on the left side of the screen for reference.

During tutoring, students are required to complete 4 phases: 1) pre-training, 2) pre-test, 3) training, and 4) post-test. In the pre-training phase, all students study the domain principles through a probability textbook by reviewing some examples and solving certain training problems. In the second phase, students take a pre-test which contains 14 problems. More specifically, the textbook is not available at this phase and students are not given feedback on their answers, nor are they allowed to go back to earlier questions. This is also true for the post-test. In phase 3, all students receive the same 12 rather complicated problems in the same order on Pyrenees tutor. Each of the 10 major principles needs to be applied at least twice in the training problems. For each problem, the average solving steps range from 20 to 50. Different from the pre- and post- test, students can access the

corresponding pre-training textbook and tutor help is available during this phase. Most importantly, the pedagogical policy works in this phase by deciding what action to take for each problem. In the training phase, each problem could have been provided as problem solving or worked example. Also, each step in the problem could have been provided as either a tell or elicit. Finally, all of the students complete a post-test with 20 problems. 14 of the problems are isomorphic to the pre-test given in phase 2. The remaining six are non-isomorphic complicated problems.

The performance of student learning is measured by the *normalized learning gain* (NLG) which is defined as $NLG = \frac{posttest - pretest}{1 - pretest}$ where 1 is the maximum score for both pre- and post- test. When grading the pre- and post- test, we use partial credit that each problem score is defined by the proportion of correct principle applications evident in the solution. For example, a student who correctly applied 4 of 5 possible principles would get a score of 0.8. All of the tests are graded in a double-blind manner by a single experienced grader. For comparison purposes, all test scores are normalized to the range of $[0, 1]$.

## 4.2 Experiment Setup

**Training Dataset** Our training dataset contains a total of 1148 students' interaction log collected over six semesters' classroom studies (16 Fall to 19 Spring). The studies were assigned as a regular homework to students. During the studies, all students used the same tutor, followed the same general procedure, studied the same training materials, and worked through the same training problems.

From the student-system interaction logs, 142 features were extracted which describes the student learning state. All the 142 features can be categorized into five groups that **Autonomy** features describe the amount of work done by the student; **Temporal** features are the time related information during tutoring; **Problem Solving** features indicate the context of the problem itself; **Performance** features denote student's performance; and **Student Action** features record the student behavior information. For each problem, there are three possible actions: worked example (WE), problem solving (PS) and step decisions (SD). In WE, the student observes how the tutor solves a problem; in PS, the student solves the problem; in SD, student solves a portion of steps in a problem while the tutor shows how to solve the others. For reward, there's no immediate reward during tutoring and the delayed reward is the student's NLG.

**Offline Learning and Evaluation** The offline learning process follows the same process with the GridWorld in section 3.2. First, NN was applied to infer the immediate rewards for the training dataset. Then, critical policy $\pi$ and $\pi_c$ were induced based on the original DQN and the Critical DQN. Finally, we fixed the threshold of ShortTRs based on the elbows in the distribution and explored the relationship between different thresholds of LongTRs and the performance of the critical policies.

Different from the online evaluation in GridWorld game, we applied *off-policy* policy evaluation (OPE) metrics to evaluate the performance of the critical policies. In general, there are two types of OPE: model based and Importance Sampling (IS) based. Song's work [12] showed that Per Decision Importance Sampling (PDIS) is the best metrics to evaluate the performance of RL-induced policies in the context of ITSs. So, PDIS was applied to evaluate the critical policies on the training dataset. More specifically, if a decision is identified as critical, the probability of taking that action is calculated by the softmax of Q-values among all the possible actions. On the contrary, if the decision is identified as non-critical, then the probability of taking that action is the random probability 1/3 as there are three possible actions for each problem.

## 4.3 Results

For Pyrenees tutor, we first present the offline evaluation results for all six critical policies. Then, we explore the identified critical decisions in the historical dataset.

### 4.3.1 Offline Evaluation Results

Figure 11 shows the offline evaluation results on Pyrenees tutor dataset. The X axis is the percentage of decisions identified as critical decisions in the historical dataset. The Y axis is the PDIS value. In general, the higher the PDIS, the better the policy.

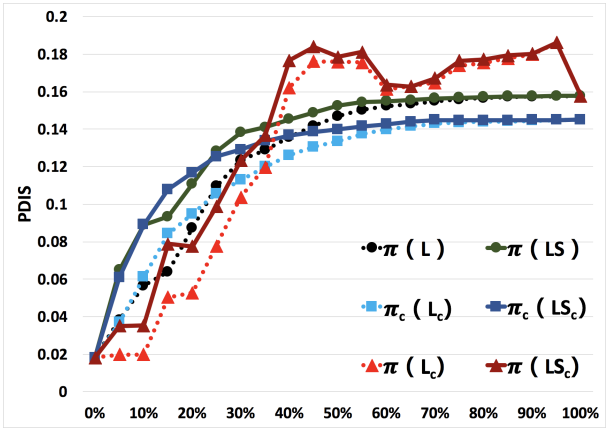


**Figure 11: Offline Evaluation Results**

First of all, the trend still holds that the more critical decisions, the better the policy would perform. When comparing within the dashed lines, there's no clear pattern before 40% threshold. However, $\pi(L_c)$ significantly outperformed the other two critical policies after 40%. The same trend occurs on the solid lines with LSTRs. The reason is that the Pyrenees dataset is not large enough for the Critical DQN to find an optimal policy, but the $L_c$ and $LS_c$ are still accurate to identify critical decisions. Furthermore, the performance jump around 50% demonstrates the reliability of the Critical DQN algorithm because in the pseudo-code 1, we already decide half of the decisions are critical decisions and the Figure 11 reflects this setting. As expected, the LSTR still outperforms LongTR that all the solid lines are above the corresponding dashed lines. In summary, the result reflects the effectiveness of LSTR in identifying critical decisions.

### 4.3.2 Exploring Critical Decisions

**Table 2: Distribution of Critical Decisions in each Problem**

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Long-Term Rewards | 3% | 16% | 16% | 13% | 15% | 11% | 7% | 7% | 6% | 6% |
| Short-Term Rewards | 0% | 1% | 6% | 6% | 6% | 10% | 13% | 19% | 19% | 20% |
| Long-Short Term Rewards | 3% | 14% | 15% | 12% | 14% | 11% | 8% | 8% | 8% | 8% |

In order to further investigate the critical decisions identified by LSTRs in the tutor dataset, we analyzed where did they occur. 50% threshold for the $LS_c$ in Figure 11 was applied to identify critical decisions in the tutor dataset and Table 2 shows the distribution of critical decisions in each problem. The first row represents the 10 problems in chronological order. The second row indicates the percentage of critical decisions identified by LongTRs in different problems. For example, 3% of critical decisions happens in P1 while 16% happens in P2. It indicates that the LongTRs focus on the critical decisions in the early to mid stage. In the meantime, the third row shows that the ShortTRs focus on the critical decisions in the late stage. The fourth row shows the critical decisions identified by LSTRs, which is the union set of the critical decisions from LongTRs and ShortTRs. Overall, critical decisions are evenly distributed among all the problems except the first one. It is not surprising that the first one is not so important because in the first problem, students are not familiar with the system and the policy needs more data to know the student status better. Furthermore, it reflects that the LongTRs and ShortTRs complement each other. If we only focus on LongTRs, we will miss the important decisions in the late stage, otherwise we will miss the important decisions in the early to mid stage.

# 5. RELATED WORK

## 5.1 RL For Pedagogical Policy Induction

Prior Research in Applying RL to Pedagogical Policy Induction can be roughly divided into classic RL vs. Deep RL approaches. The latter is highly motivated by the fact that the combination of deep learning (neural networks) and novel reinforcement learning algorithms has made solving complex problems possible in the last decade. For instance, the Deep Q-Network (DQN) algorithm [18] takes advantage of convolutional neural networks to learn to play Atari games observing the pixels directly. Since then, DRL has achieved success in various complex tasks such as the games of Go [25], Chess/Shogi [26], and robotic control [3]. One major challenge of these methods is *sample inefficiency* where RL policies need large sample sizes to learn optimal, generalizable policies. Batch RL, a sub-field of RL, aims to fix this problem by learning the optimal policy from a fixed set of a priori-known transition samples [15], thus efficiently learning from a potentially small amount of data and being able to generalize to unseen scenarios.

Prior research using classic RL approaches has applied both online and batch/offline approaches to induce pedagogical policies for ITSs. Beck et al. [4] applied temporal difference learning to induce pedagogical policies that would minimize the students' time on task. Similarly, Iglesias et al. applied Q-learning to induce policies for efficient learning [10]. More recently, Rafferty et al. applied an online partially observable Markov decision process (POMDP) to induce policies for faster learning [19]. All of the models described above were evaluated via simulations or classroom studies, yielding improved student learning and/or behaviors as compared to some baseline policies. Offline or batch RL approaches, on the other hand, "take advantage of previous collected samples, and generally provide robust convergence guarantees" [22]. Thus, the success of these approaches depends heavily on the quality of the training data. One common convention for collecting an exploratory corpus is to train students on ITSs using *random yet reasonable* policies. Shen et al. applied value iteration and least square policy iteration on a pre-collected exploratory corpus to induce a pedagogical policy that improved students' learning performance [24, 23]. Chi et al. applied policy iteration to induce a pedagogical policy aimed at improving students' learning gain [5]. Mandel et al. [16] applied an offline POMDP to induce a policy which aims to improve student performance in an educational game. All the models described above were evaluated in classroom studies and were found to yield certain improved student learning or performance relative to a baseline policy. Wang et al. applied an online DRL approach to induce a policy for adaptive narrative generation in educational game using simulations [29]; the resulting DRL-induced policies were evaluated via simulations only. In this work, based on the characteristics of our task domain, we focus on batch RL with neural networks, also known as batch Deep Reinforcement Learning (batch DRL) [11, 9].

## 5.2 Critical Decisions in Simulation

Student-Teacher framework is the most closely related work to our problem. In this framework, a "student" agent learns from the interaction with environment, while a "teacher" agent provides action suggestions to accelerate the learning process. Their research question is not what to advise but when to advise, especially with a limited budget of advice.

Clouse [7] was the first one studied the student-teacher framework in a student-initiated advising mode. They applied Q-value difference to measure the student's confidence in a state and used it to decide when should the student ask for help. The results showed that compared with random asking, their approach could improve the learning speed significantly. Furthermore, the experiment demonstrated that not all the teacher's advice are equally helpful. The same amount of advice can cause the student agent to take widely varying amounts of steps to find the optimal policy.

Torrey et al. [28] considered the student-teacher framework in teacher-initiated advising way. They considered an environment with a limited budget of advice and teacher decided when to give advice. They proposed several heuristic methods to determine when to give advice such as early advising, importance advising, mistake correcting and predictive advising. The results showed that mistake correcting has the best performance which indicates that advice can have the greatest impact when students make mistakes on important

states.

Zimmer et al. [33] modeled the when to advise problem as an RL problem. They learned a teaching policy with two actions: $A = advice, noadvice$ to decide when to give advice to the student. Compared with heuristic methods, the result showed that the teacher policy is effective because it can learn not only when to give advice, but also distinguish good and bad student agent that good agent chooses a lot of good actions and doesn't need advice while bad agent needs more.

Amir et al. [1] studied the jointly-initiated strategies for student-teacher learning framework. In their model, both student and teacher can initiate advising based on heuristic functions. The motivation of their work is to reduce the pressure of the teacher agent on monitoring the student constantly and make the framework more close to the real-life student-agent scenario. The result showed that the joint decision-making approach could reduce the attentions required from the teacher but still keep the student learning effectively.

Fachantidis et al. [8] explored the impact of advice quality in the student-teacher framework. They distinguished teacher agents to be an expert or a good teacher who provide optimal or sub-optimal advice. Also, a Q-teaching method was proposed to learn a teaching policy to decide when to give advice. Their results showed that the best performers are not always the best teachers and the Q-teaching approach is significantly more efficient than others.

In summary, prior works investigated the problem of when to give advice in simulated environments. They showed that Q-value difference is a robust and accurate heuristic function to estimate the importance of decision in interactive environments. However, prior works only considered RL-based student agent but not human students. In this work, we expand to a dataset of real-world ITS involving human students.

## 5.3 Exploiting Q-value Difference in ITSs

Some prior work exploited the Q-value difference between actions to simplify the decision-making process/problem in the context of ITS. For example Mitchell et al. relied on the Q-value difference to reduce the feature space [17]. More specifically, they proposed a policy evaluation metric, separation ratio for feature selection, which is defined as $\frac{2*|Q(s,a_1)-Q(s,a_2)|}{(Q(s,a_1)+Q(s,a_2))}$ where $Q(s, a_i)$ is the Q value for the state-action pair $(s, a_i)$. The feature selection approach was then combined with RL to induce pedagogical policies for a dialog system, the Java tutor.

Zhou et al. [31] relied on Q-value difference to reduce the policy space. More specifically, they applied weighted decision tree with post-pruning to extract a compact set of 529 rules from a full set of 3706 rules. During the extraction, each rule was weighted by the Q-value difference between two alternative actions and thus increased the carry-out likelihood of more important decisions. The policies were empirically evaluated in a classroom study. Results showed that the full RL policy and the compact DT policy together were significantly more effective than a random policy and there is no significantly difference between the full RL policy and the compact DT policy.

Song et al. [13] proposed an ADRL framework to identify critical decisions and conducted an empirical study to test the effectiveness of the ADRL. In ADRL, two policies were induced that a positive policy aims to help student while a bad policy tries to hinder student learning. For a given state, if the two policies have different decisions and the Q-value difference is large enough, then this is a critical state and the decision is important. The results showed that critical phase exists in student learning that critical decisions always occur in groups and the more critical phase students have experienced, the better performance they have.

In sum, prior studies have considered the Q-value difference between actions as a heuristic function of action importance. The larger the difference, the more important the decision is. However, prior work didn't quantitatively study how large Q-value difference is a critical decision. In this work, we explored the Q-value difference thresholds by classifying decisions into two categories: critical and non-critical and evaluating the quality of the critical decisions.

# 6. CONCLUSIONS

In this study, we explored Long-Short Term Rewards to identify critical decisions in both synthetic Gridworld game and real-world ITS. Based on the LSTRs, we proposed Critical DQN to induce critical policy whose Q-value difference is more heuristic and sensitive to the decision importance. In order to investigate the effectiveness of LSTRs, we evaluated the performance of critical policies with different critical thresholds by online evaluation on GridWorld and offline evaluation on Pyrenees tutor's dataset. The results showed that the LongTRs from Critical DQN are significantly better than the original DQN. Through considering the ShortTRs, the LSTRs are significantly better than the LongTRs. However, the Critical DQN needs more data to converge to an optimal policy. In summary, through identifying critical decisions by the LSTRs, half of the decisions are trivial and carry out the optimal policy on the 50% decisions (critical ones) could achieve the similar effect of carrying out on all decisions.

In the future, we plan to generalize the LSTR framework to other domains in terms of interactive environments. Also, we hope to deploy a critical DQN policy on Pyrenees tutor and only carry out 50% decisions in a classroom study to test the critical decisions empirically.

# 7. ACKNOWLEDGEMENT

This research was supported by the NSF Grants: CAREER: Improving Adaptive Decision Making in Interactive Learning Environments(1651909), Integrated Data-driven Technologies for Individualized Instruction in STEM Learning Environments(1726550), and Generalizing Data-Driven Technologies to Improve Individualized STEM Instruction by Intelligent Tutors (2013502).

# 8. REFERENCES

[1] O. Amir, E. Kamar, A. Kolobov, and B. J. Grosz. Interactive teaching strategies for agent training. *the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI*, pages 804–811, 2016.

[2] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier. Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2):167–207, 1995.

[3] M. Andrychowicz, B. Baker, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[4] J. Beck, B. P. Woolf, and C. R. Beal. Advisor: A machine learning architecture for intelligent tutor construction. *AAAI/IAAI*, 2000(552-557):1–2, 2000.

[5] M. Chi, K. VanLehn, D. Litman, and P. Jordan. Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. *User Modeling and User-Adapted Interaction*, 21(1-2):137–180, 2011.

[6] B. Clement, P.-Y. Oudeyer, and M. Lopes. A comparison of automatic teaching strategies for heterogeneous student populations. In *EDM*, 2016.

[7] J. A. Clouse. On integrating apprentice learning and reinforcement learning. *PhD thesis*, 1996.

[8] A. Fachantidis, M. E. Taylor, and I. P. Vlahavas. Learning to teach reinforcement learning agents. *Machine Learning and Knowledge Extraction*, 2017.

[9] S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

[10] A. Iglesias, P. Martínez, R. Aler, and F. Fernández. Reinforcement learning of pedagogical policies in adaptive and intelligent educational systems. *Knowledge-Based Systems*, 22(4):266–270, 2009.

[11] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[12] S. Ju, S. Shen, H. Azizsoltani, T. Barnes, and M. Chi. Importance sampling to identify empirically valid policies and their critical decisions. *EDM Workshop*, 2019.

[13] S. Ju, G. Zhou, H. Azizsoltani, T. Barnes, and M. Chi. Identifying critical pedagogical decisions through adversarial deep reinforcement learning. *EDM Poster*, 2019.

[14] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark. Intelligent tutoring goes to school in the big city. 1997.

[15] S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[16] T. Mandel, Y.-E. Liu, S. Levine, E. Brunskill, and Z. Popovic. Offline policy evaluation across representations with applications to educational games. In *AAMAS*, pages 1077–1084, 2014.

[17] C. M. Mitchell, K. E. Boyer, and J. C. Lester. Evaluating state representations for reinforcement learning of turn-taking policies in tutorial dialogue christopher. *SIGDIAL*, pages 339–343, 2013.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[19] A. N. Rafferty, E. Brunskill, T. L. Griffiths, and P. Shafto. Faster teaching via pomdp planning. *Cognitive science*, 40(6):1290–1332, 2016.

[20] J. Rowe, B. Mott, and J. Lester. Optimizing player experience in interactive narrative planning: a modular reinforcement learning approach. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[21] J. P. Rowe and J. C. Lester. Improving student problem solving in narrative-centered learning environments: A modular reinforcement learning framework. In *AIED*, pages 419–428. Springer, 2015.

[22] D. Schwab and S. Ray. Offline reinforcement learning with task hierarchies. *Machine Learning*, 106(9-10):1569–1598, 2017.

[23] S. Shen and M. Chi. Aim low: Correlation-based feature selection for model-based reinforcement learning. In *EDM*, pages 507–512, 2016.

[24] S. Shen and M. Chi. Reinforcement learning: the sooner the better, or the later the better? In *the 2016 Conference on User Modeling Adaptation and Personalization*, pages 37–44. ACM, 2016.

[25] D. Silver, A. Huang, C. J. Maddison, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[26] D. Silver, T. Hubert, J. Schrittwieser, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[27] J. C. Stamper, M. Eagle, T. Barnes, and M. Croy. Experimental evaluation of automatic hint generation for a logic tutor. In *AIED*, pages 345–352. Springer, 2011.

[28] L. Torrey and M. E. Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13*, pages 1053–1060, 2013.

[29] P. Wang, J. Rowe, W. Min, B. Mott, and J. Lester. Interactive narrative personalization with deep reinforcement learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[30] G. Zhou, H. Azizsoltani, M. S. Ausin, T. Barnes, and M. Chi. Hierarchical reinforcement learning for pedagogical policy induction. In *Artificial Intelligence in Education - 20th International Conference, AIED 2019, Chicago, IL, USA, June 25-29, 2019, Proceedings, Part I*, pages 544–556. Springer, 2019.

[31] G. Zhou, J. Wang, C. Lynch, and M. Chi. Towards closing the loop: Bridging machine-induced pedagogical policies to learning theories. In *EDM*, 2017.

[32] G. Zhou, X. Yang, H. Azizsoltani, T. Barnes, and M. Chi. Improving student-tutor interaction through data-driven explanation of hierarchical reinforcement induced pedagogical policies. In *UMAP*. ACM, 2020.

[33] M. Zimmer, P. Viappiani, and P. Weng. Teacher-student framework: A reinforcement learning approach. *AAMAS Workshop Autonomous Robots and Multirobot Systems*, 2013.