# Deep Reinforcement Learning to Simulate, Train, and Evaluate Instructional Sequencing Policies

Jithendaraa Subramanian
National Institute of Technology, Tiruchirappalli
Tamil Nadu, India
jithen.subra@gmail.com

Jack Mostow
Carnegie Mellon University
Pittsburgh, PA 15213
mostow@cs.cmu.edu

## ABSTRACT

Instructional sequencing in Intelligent Tutoring Systems (ITS) refers to the pedagogical policy governing tutor decisions such as choosing the next topic, activity, problem, or feedback to help users learn. This policy affects students' learning gains, engagement, and efficiency (speed of learning). We used reinforcement learning to train pedagogical policies for an ITS, with three novel aspects compared to prior work in this area. First, we used a different student modeling method, HOT-DINA, previously shown to achieve higher predictive accuracy than conventional Bayesian Knowledge Tracing. Second, we used deep reinforcement learning, which has had limited exploration for this purpose. Specifically, we employed Deep-Q-networks (DQN), asynchronous actor-critic (A2C), and Proximal Policy Optimization (PPO) to learn three pedagogical policies for optimizing expected learning gains. Third, we evaluated these policies on simulated students trained on longitudinal log data from hundreds of children using RoboTutor, a tablet tutor for basic Swahili literacy and numeracy. We evaluated the trained policies by estimating the simulated students' learning gains. They were higher than the actual students' learning gains inferred from the historical data, and higher than the simulated students' learning gains from using the original policy.

## Keywords

Intelligent Tutoring Systems, Deep Reinforcement Learning, Adaptive Instructional Sequencing

## 1. INTRODUCTION

An Intelligent Tutoring System (ITS) aims to teach a set of target knowledge components such as concepts or skills. Teaching involves sequential decisions at various grain sizes, such as what knowledge to teach, what activities to provide, what problems to present, and what help to give. These instructional sequencing decisions affect student learning [14], so improving them can make an ITS more effective.

We study this problem in the context of RoboTutor, a $1M Finalist in the Global Learning XPRIZE Competition to develop an open source Android tablet tutor to teach basic Swahili literacy and numeracy to Tanzanian children ages 7-10 without requiring adult intervention (www.robotutor.org). XPRIZE independently field-tested RoboTutor for 15 months in 28 villages in Tanzania. The data for this paper comes from the version of RoboTutor used during the last 3 months of the XPRIZE field evaluation.

Advances in computing speed, memory size, data storage, cognitive models of student learning [24, 23], availability of large data sets, and deep reinforcement learning (RL) [13, 12, 15] have made the use of RL for adaptive instructional sequencing very promising. Deep RL has had some exploration, albeit limited, in learning teaching policies for an ITS. By using deep RL, we aim to discover better teaching policies.

Ideally, one would evaluate a teaching policy by using it with actual students and measuring its impact on their learning gains from pre to posttest. In practice, such evaluations require long studies, large sample sizes, and costly resources to demonstrate significant effects. Instead, we use data logged by an ITS to train a student model, and use it to simulate the policy's impact on student learning. By using different deep RL methods to discover policies, we can determine which ones achieve the highest (simulated) learning gains.

To find an optimal decision policy, RL searches a space of action sequences by a simulated agent, computing the effect of each action, and using it as feedback on earlier decisions to arrive at an optimal policy for what action to choose in each state. For instructional sequencing, we use a simulated tutor as the RL agent, and a simulated student to estimate the effects of its actions.

The rest of the paper is organized as follows. Section 2 relates this paper to prior research. Section 3 describes the student and tutor models used by our simulated agent. Section 4 describes the methods we use to learn decision policies. Section 5 describes how we evaluate the learned policies. Section 6 concludes and discusses limitations of this work.

## 2. RELATION TO PRIOR WORK

Instructional sequencing was first formulated as an RL problem as early as 1960 [9], though the term "reinforcement

learning" was not used until much later. Work on learning policies for instructional sequencing, reviewed in [7], has varied in terms of optimization criterion, student model, learning method, and comparison baseline, as we now summarize.

**Optimization criteria:** Past work [2, 16, 21] focused on optimizing efficiency, specifically minimizing students' time per problem, and consequently their total learning time. We focus instead on optimizing learning gains, i.e. how much the student learns.

Whitehill and Movellan [21] worked on optimizing the time for language learners to learn vocabulary (e.g. "man," "salad") from illustrations (e.g. a picture of a man eating a salad). They used a novel Bayesian student model [19] and posed the learning task as a discrete time, continuous-state, discrete-action Partially Observable Markov Decision Process problem. They used the REINFORCE policy gradient technique [22] to optimize time per problem. The learned policy taught students 27% faster, but did not significantly increase learning gains.

**Student models:** Work on optimizing instructional sequencing has used different student models and learning methods. Beck et al. [2] used log data from the AnimalWatch tutor to fit a linear Population Student Model, and Temporal-Difference Learning [18] to learn a teaching policy that optimized time per problem. Compared to a heuristic baseline, the learned policy reduced the time per problem by 30%.

**Learning methods:** Shen et al. [16] used a Constrained-Action Partially Observable Markov Decision Process where the actions modeled the tutor and their effects modeled the student. They used least squares temporal difference Q learning [10] with different objective functions to optimize either learning gains (beating a random baseline, but only for low competence students) or time per problem (beating a DQN baseline).

**Baselines:** Chi et al. [4] used data from an ITS based on a random-choice policy to train delayed-reward RL policies that proved significantly more effective.

Shen et al. [17] performed a between-groups experiment to compare reinforcement learning with immediate vs. delayed rewards for a tutor choosing between two actions, namely giving a worked example or just a problem statement. Students taught using a policy that optimized expected cumulative reward averaged faster posttest performance than students taught using a policy that optimized immediate reward.

David et al. [6] used Bayesian Knowledge Tracing (BKT) to model students and select problems in real time in the classroom. They compared this sequencing algorithm to a baseline sequencing approach that sampled problems of varying difficulty levels according to a domain expert. Students using the baseline policy performed better on easier problems while students using the BKT-based sequencing algorithms performed better on harder problems. A classroom survey showed that the students preferred the BKT-based sequence over the baseline sequence.

# 3. MODELING A STUDENT AND SIMULATING ROBOTUTOR

The student simulator should behave like students who use RoboTutor. Accordingly, the simulator uses a student model trained on logged data. It assumes that a student at step t is in a binary knowledge state *know* for each skill - the student either knows the skill or does not. However, knowledge states are hidden, so the tutor simulator cannot directly observe them. Instead, it observes the student's performance on each attempt to use a given skill. Moreover, student performance is a noisy function of student knowledge – a student may slip in applying a known skill, or accidentally apply an unknown skill correctly by guessing the answer. So our student model represents just the probability $\Pr(Know_t)$ that the student knows the skill at time step t, based on the student's observed performance.

Simulating RoboTutor is much simpler. For the work reported here, we just need to represent the tutor state that affects the behavior of the simulated student. Therefore the tutor state captures only the essentials, such as which problem to present next, and when to promote (or demote) the student.

## 3.1 HOT-DINA Student Model

To train our student model, we use HOT-DINA, a hybrid of Bayesian Knowledge Tracing (BKT) [24] and Item Response Theory (IRT) [8], because it achieved significantly higher predictive accuracy than BKT on a widely used data set from an algebra tutor [23].

Student models in intelligent tutors typically use Bayesian Knowledge Tracing (BKT) to estimate students' knowledge from their observed performance. A BKT model [5] has 5 parameters: *knew*, *learn*, *forget*, *slip*, and *guess*. The *knew* parameter is the probability that the student knows the skill prior to practising it. The *learn* parameter is the conditional probability of knowing the skill at step t+1 given that the student does not know it at step t. Conversely, the *forget* parameter is the conditional probability of not knowing the skill at step t+1 given that the student *did* know it at step t. The *guess* parameter is the conditional probability of an attempt succeeding given that the student does not know the skill. Conversely, the *slip* parameter is the conditional probability of an unsuccessful attempt given that the student actually knows the skill. Typically *forget* is assumed to be 0, and the model parameters (*knew*, *learn*, *guess*, and *slip*) are fit to students' logged performance data using the Expectation Maximization algorithm [20].

HOT-DINA has the same *guess*, *slip*, and *learn* parameters as BKT, but estimates its *knew* parameter based on these parameters from IRT:
$\theta_n$: Proficiency or ability of student n
$a_k$: Discrimination of skill k
$b_k$: Difficulty of skill k
$\alpha_{nk}^{(0)}$: Pr(student n knew skill k before starting practice)

$$\alpha_{nk}^{(0)} = \frac{1}{1 + exp(-1.7a_k(\theta_n - b_k))} \quad (1)$$

Thus, in estimating $knew_{nk} = \alpha_{nk}^{(0)}$, HOT-DINA extrapolates from student $n$'s knowledge of other skills, and from

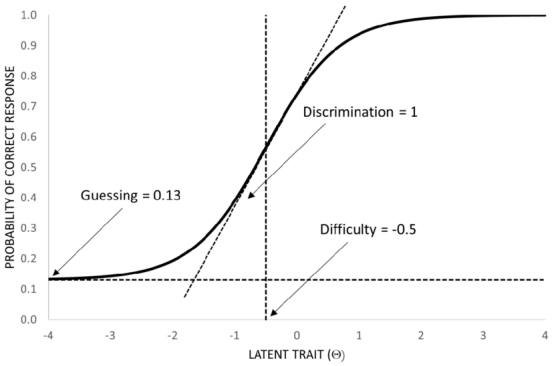


**Figure 1: Pr(Correct) vs. student ability in an IRT model [3]**

other students' knowledge of skill $k$, albeit at a high computational cost to fit the model. The *knew* parameter from HOT-DINA varies with student proficiency in the same way that Pr(Correct) varies with latent student ability in an IRT 3PL model, as illustrated in figure 1.

HOT-DINA computes $\alpha_{nk}^{(t)} = \Pr(\text{student } n \text{ knows skill } k \text{ at time } t)$ for subsequent time steps $t$ by updating it in the same way as Bayesian Knowledge Tracing. Following the BKT updates, HOT-DINA updates student knowledge as explained below. The next few lines cover some basic notation and the update equations for the HOT-DINA student model. Note that variables $q$, $\alpha$, $y$, and $Y$ are all binary, but we use (.) as shorthand for Pr(.).
$(\alpha_{nk}^{(t)} = 1)$: Pr(student $n$ knows skill $k$ at step $t$)
$(y_{nk}^{(t)} = 1)$: Pr(student $n$ applies skill $k$ correctly at step $t$)
$(Y_{nj}^{(t)} = 1)$: Pr(student $n$ succeeds on an attempt at activity $j$ at step $t$)
$q_{jk}$: 1 if attempt $j$ exercises skill $k$, otherwise 0

$$(y_{nk}^{(t)} = 1) = (1 - slip_k)(\alpha_{nk}^{(t)} = 1) + guess_k(\alpha_{nk}^{(t)} = 0) \quad (2)$$

$$(Y_{nj}^{(t)} = 1) = \prod_{k=1}^{K} (y_{nk}^{(t)} = 1)^{q_{jk}} \quad (3)$$

$$(\alpha_{nk}^{(t)} = 1 | Y_{nj}^{(t)} = 1) = (\alpha_{nk}^{(t)} = 1) * \left( \frac{1 - slip_k}{Y_{nj}^{(t)} = 1} \right)^{q_{jk}} \quad (4)$$

$$(\alpha_{nk}^{(t)} = 1 | Y_{nj}^{(t)} = 0) = (\alpha_{nk}^{(t)} = 1) * \left( \frac{slip_k}{1 - (Y_{nj}^{(t)} = 1)} \right)^{q_{jk}} \quad (5)$$

$$(\alpha_{nk}^{(t+1)} = 1) = (\alpha_{nk}^{(t)} = 1 | Y_{nj}^{(t)}) + (learn_k * (\alpha_{nk}^{(t)} = 0 | Y_{nj}^{(t)})) \quad (6)$$

## 3.2 Simulating a Student

The student model must perform two functions. First, it must simulate whether an attempt to use a given skill succeeds, given the probability that the student knows the skill. Second, based on the outcome, it must update its estimated probability that the student knows the skill. To simulate a student's performance, we first estimate Pr(Attempt $j$ succeeds) as in equations 2 and 3. We then simulate the student attempt (successful or unsuccessful) by doing a biased coin flip based on this estimated probability. Given the simulated outcome, we perform knowledge tracing over multiple skills using the update equations 3-6.

**Table 1: Converged individual proficiency parameters for the HOT-DINA student model.**

| Parameter | Mean | Std. deviation |
|---|---|---|
| $\theta_1$ | -0.17 | 0.93 |
| $\theta_2$ | 1.77 | 0.64 |
| $\theta_3$ | -0.65 | 0.72 |
| $\theta_4$ | -0.73 | 1.13 |
| $\theta_5$ | -0.38 | 0.78 |
| $\theta_6$ | -0.28 | 0.55 |
| $\theta_7$ | -0.48 | 0.7 |
| $\theta_8$ | 0.14 | 0.47 |

## 3.3 Training the Student Model

We have already defined our student model HOT-DINA To limit computation time, we fit the model on logged data from a single village, consisting of 42,010 attempts by 8 children to apply 22 skills. We fit one proficiency parameter for each child and 5 parameters for each skill (*guess*, *slip*, *learn*, *difficulty*, and *discrimination*), 118 parameters in total. Fitting 5 separate parameters per activity instead of per skill might achieve higher accuracy but would require fitting 8,558 parameters. The original HOT-DINA used OpenBUGS Gibbs sampling package [11]. However, we use MCMC sampling [1] for Bayesian inference with PyStan (a statistical modeling tool) because it is faster than OpenBUGS and handles larger datasets. Fitting the 118-parameter HOT-DINA model to 42,010 attempts took approximately 4 days on a supercomputer with 128 GB RAM and 28 cores.

Table 1 shows the converged $\theta$ values for the 8 individual student proficiency parameters of the student model. Space precludes listing the values of the 110 *guess*, *slip*, *learn*, *difficulty*, and *discrimination* parameters estimated for the 22 skills. Once we obtain the model parameters, we can simulate whether an attempt succeeds, and based on this outcome, perform knowledge tracing over multiple skills to update the student's knowledge probabilities.

## 3.4 Simulating RoboTutor

A session starts when the student launches RoboTutor and ends when the student exits. RoboTutor has a total of 1710 educational activities in three content areas – literacy, numeracy, and stories, each with its own sequence of activities. Each activity gives assisted practice on one or more skills using a sequence of items (typically 10), such as letters or words to write, number problems to solve, or sentences to read. Each item requires one or more steps and each step may take the student one or more attempts.

RoboTutor rotates through the three content areas, keeping track of the student's position $i$ in each content area. The student can repeat the same activity, choose the activity at position $i$ or $i+1$ in the next content area in the rotation, or exit RoboTutor. Thus the tutor state specifies the current content area and the student's position in it. Based on the student's percentage of successful attempts during the ac-

tivity, RoboTutor either demotes the student BACK to the previous position $i-1$ in the current content area, stays at the SAME position $i$, advances to the NEXT position $i+1$, or SKIPs to position $i+2$.

Our student and tutor simulator deviate from RoboTutor in a few respects. First, RoboTutor lets the student repeat the previous activity or choose between activities $i$ and $i+1$. In contrast, the simulated student always chooses activity $i$, i.e. we do not model student preference. Second, RoboTutor lets the student attempt all the items in the current activity or quit at any point to choose another activity. In contrast, the simulated student chooses activities one item at a time, i.e., we do not model student persistence. Third, if the student makes less than 3 attempts before quitting an activity, RoboTutor moves to either position $i-1$ or $i+1$ with equal probability, on the grounds that the number of attempts is insufficient to assess, but the student does not want to repeat the activity at position $i$. Fourth, RoboTutor chooses among BACK, SAME, NEXT, and SKIP based on fixed heuristic thresholds on the percentage of correct attempts. In contrast, the simulated tutor chooses among these 4 actions based on the learned policy.

# 4. LEARNING INSTRUCTIONAL POLICIES

We aim to use deep RL methods to learn a teaching policy that maximizes long-term rewards formulated as the learning gains (LG). The goal of an RL agent is to learn a policy $\pi$, defined as a mapping from state space $S$ to action space $A$. Given any state, the RL agent follows a series of actions chosen by the learned policy to maximize the long-term expected reward. In the context of an ITS, we specify the RL agent as follows:

**State** $s_t$: We define the state as a combination of the student state and the tutor state. The tutor state determines the set of actions available to the RL agent at a given step. We represent the student state as a vector of probabilities where element $k$ is the estimated probability that the student knows skill $k$.

**Action** $a_t$: The actions taken by the RL agent correspond to making tutorial promotion decisions: BACK, SAME, NEXT, or SKIP.

**Next state** $s_{t+1}$: The knowledge vector of a student after a Bayesian update based on the simulated student's response to tutor action $a_t$ in state $s_t$ is the updated student knowledge state. The updated tutor state is given by the tutor simulator. The updated student knowledge state and tutor state together give the next state $s_{t+1}$.

**Reward** $r_t(s_t, a_t)$: We use a novel reward function defined as the average difference between prior and posterior knowledge states based on the simulated student's response to the tutor action $a_t$ to the student simulator as defined in equation 7. Our goal is to maximize the long-term rewards (LG) with time horizon T (equation 8).

$$r_t(s_t, a_t) = \frac{\sum_{k=1}^{K} (\alpha_{nk}^{(t+1)} = 1) - (\alpha_{nk}^{(t)} = 1)}{K} \tag{7}$$

$$LG = \sum_{t=1}^{T} r_t(s_t, a_t) = \frac{\sum_{k=1}^{K} (\alpha_{nk}^{(T)} = 1) - (\alpha_{nk}^{(0)} = 1)}{K} \tag{8}$$

We employ Deep RL methods - DQN, A2C, and PPO - to learn teaching policies by simulating tutor-student interactions. We use a novel state representation consisting of a student state and a tutor state. We represent student $n$'s knowledge state as the vector of estimated probabilities of student n knowing skill $k$ as $[(\alpha_{n1}^{(t)} = 1), ..., (\alpha_{nK}^{(t)} = 1)]$. The tutor state includes the current content area (literacy, numeracy, or stories) and the student's current position in the curricular sequence for that area. We use a novel reward function formulated as the estimated learning gains: average difference (across 22 skills) in student knowledge after T=500 steps of problem solving. Our actions consist of choosing BACK, SAME, NEXT, or SKIP while following the content area rotation constraints. Figure 2 shows a diagrammatic overview of our framework, STEP: Simulate, Train, Evaluate Policy.

We learned a decision policy – a mapping from states to actions – that maximizes the total expected reward of following the policy $\pi_\theta$. As the reward function for student $n$, we use the knowledge gain estimated by the student model, i.e. posterior minus prior estimates of Pr(student $n$ knows skill $k$), averaged over all skills. The total reward (called "returns") is just the cumulative rewards over 500 steps. The posterior and prior refer to the estimated knowledge states before and after applying the Bayesian updates (equations 4-6) on an activity selected by tutor action $a_t$. The information for prior knowledge is implicitly present in the knowledge state $s_t$. In order to save computational time, we learned policies with a time horizon of $T$=500 steps. Though our experiments stick to a finite horizon and undiscounted returns, it is possible to extend this approach to any finite number of steps or even to infinite-horizon discounted returns with discount factor $\gamma \in (0, 1)$.

For every RL algorithm in our study (DQN, A2C, and PPO), we learn exactly one policy assuming we have to teach a simulated student with very low knowledge ($\alpha_{nk}^0$ = Pr(knew) = 0.05 for all 22 skills). The reason for this assumption is so that our RL agents can explore larger areas of the state space. For example, if we were to learn a teaching policy for a student with a prior Pr($know_j$)=0.5 for $j$=1..22, the agent could explore only parts of the knowledge state space $\geq 0.5$ (since we assume the student does not forget knowledge) and thus would not be able to generalize to students with low knowledge ($< 0.5$). On the other hand, if we train on a student with very low knowledge (=0.05), this student will eventually be able to reach a higher knowledge state and the policy will still be able to take optimal actions since this state space has been explored before.

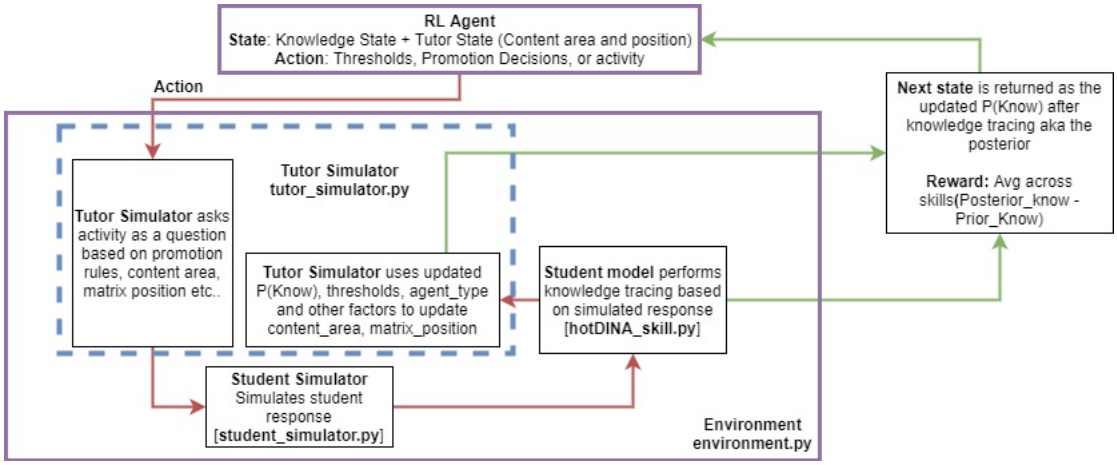


**Figure 2: A diagrammatic overview of STEP: our RL architecture**

**Table 2: Mean % improvement of deep RL over baselines**

| Deep RL method | Original policy | Random baseline |
|---|---|---|
| DQN | 2.18% | 0.94% |
| A2C | 5.45% | 4.14% |
| **PPO** | **5.48%** | **4.18%** |

## 5. EVALUATION AND RESULTS

We evaluated our learned RL policies against a historical baseline and two baseline policies – a random policy and the policy used in the original tutor – across 8 students from a village that used RoboTutor. It is important to note that a random policy is not necessarily bad. For example, all actions in an ITS might improve student learning (but by various amounts), in which case a random policy can be a decent baseline [16, 25]. For each student, we evaluate 3 different teaching policies, induced by the DQN, A2C, and PPO algorithms.

Table 2 compares the percentage improvement in learning gains as a result of using deep RL methods over both baselines. Percent improvement was calculated as the fraction of the difference in learning gains between baseline and proposed policy and the baseline policy. The *original policy baseline* is based on the simulated tutor using the original promotion policy and simulated students trained on actual students' historical data. The original policy compared a student's percentage of correct attempts against thresholds based on which the policy could either demote a student BACK to a previous activity, retain the student on the SAME activity, promote the student to the NEXT activity, or advance the student to the activity thereafter (SKIP). The *random policy baseline* randomly chose BACK, SAME, NEXT, or SKIP.

Figure 3 shows how student knowledge, averaged across all 22 skills, evolves over 500 time steps – i.e. the first 500 attempts. The figure displays three baselines as dashed lines:

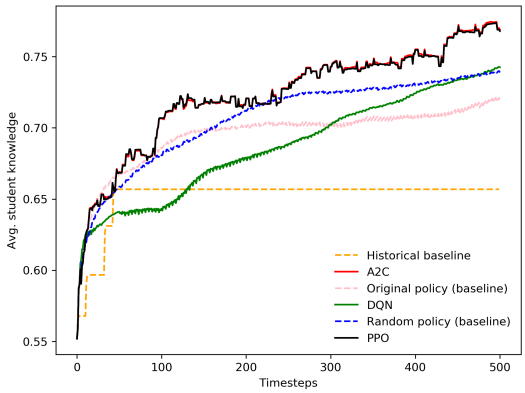


**Figure 3: Learning gains of deep RL policies vs. baseline policies for one of the 8 students**

- *Historical baseline* for one of the 8 students, estimated by the student model from the student's 734 logged attempts. (All 8 students made more than 500 attempts.) Much of this learning curve looks flat because it's averaged over 22 skills of which only 1 or 2 changed at each time step. RoboTutor continued each activity until the student completed or exited it, which the student did much less often after successful attempts than after unsuccessful attempts. Consequently, the estimated probability of knowing a skill tended to reach ceiling early in an activity and then stay flat until starting a new activity, resulting in a terraced learning curve – so much so that one reviewer even asked if there was any learning at all after the first 75 time steps.

- *Original policy* evaluated on simulation of the same student driven by a model fit to the student's estimated proficiency, with its 110 skill parameters trained on data from all 8 students. Learning curves for the simulated student were averaged over 500 Monte Carlo runs. The simulated tutor was free after each attempt to switch to a different activity, where knowledge gain might be higher, in contrast to the historical baseline, which changed activities only 16 times. Consequently rather than matching the historical baseline, the simulated student's learning curve was much faster.

- *Random policy* applied to this simulated student.

The solid lines show the same simulated student's learning curves under the policies produced by the 3 deep RL methods:

- *DQN policy* learned by the Deep Q Network method
- *A2C policy* learned by the Advantage Actor Critic method
- *PPO policy* learned by Proximal Policy Optimization

Figure 3 shows learning curves for a single student, but all 3 deep RL methods beat the baseline learning gains for each of the 8 students, except for two where DQN performed comparably to the baseline policies. As Figure 3 illustrates, the A2C and PPO policies resulted in nearly identical learning curves even though A2C and PPO were trained separately, suggesting that they discovered optimal teaching policies but DQN did not. Even though averaged over 500 Monte Carlo runs, A2C and PPO fluctuate up and down, in contrast to the smooth shape of the other learning curves. As an insightful reviewer pointed out, this fluctuation suggests that A2C and PPO may be "taking more adventurous actions compared to other policies," leading to higher learning gains at the cost of "some very challenging experiences for the simulated students" along the way.

# 6. CONCLUSION: CONTRIBUTIONS, LIMITATIONS, AND FUTURE WORK

This paper introduces our STEP framework (Simulate, Train, Evaluate Policy), which uses deep RL to maximize learning gains in an ITS and discovered policies that beat baseline policies. To the best of our knowledge, this is the first work that compared different deep RL methods for maximizing ITS learning gains by simulated students trained on data from real students. Most prior work either used deep RL to optimize problem solving time or did not outperform baselines. STEP is a novel combination of the HOT-DINA student model with deep RL for policy learning and uses a simulated student trained on historical data.

We used a novel state representation consisting of a student state – formulated as the vector of knowledge probabilities per skill – and tutor state. We used simulated learning gains as a new reward function, computed as the average change in estimated knowledge probability during a student-tutor interaction. Though all 3 deep RL methods found good teaching policies, PPO was the most promising; A2C learned a policy close to that of PPO; and DQN did not perform as well as PPO or A2C, but still beat the baseline policies. This result supports the use of deep RL induced policies for instructional sequencing in an ITS.

We trained a single student model on a set of 8 students, estimating proficiency for each student, and other parameters for each skill. The resulting activity selection policy could be used with unseen students to adapt to their performance by assuming average proficiency, in contrast to individualization methods that must be trained on data from the individual. The learned policy is student independent in that we evaluated a single learned policy on all 8 students in our study. We focus here on a particular level of policy learning, namely activity selection, ignoring item-level effects that we would have to model in order to learn item selection policies. RoboTutor's behavior at the grain size of activities is easy to model because it follows a fixed rotation among content areas and a simple policy for promotion based on performance. Modeling its behavior at finer grain sizes would depend on details of specific activities.

Our work has several limitations. Most obviously, rather than evaluate on actual students, we used a model-based approach to simulate students. Any policy based on a student model might not be as good as it seems. Although we trained the model on historical data from real students, this approach is limited not just by the amount of data collected but by the model's assumptions about student learning. Future work should explore better student modeling methods by relaxing these assumptions.

One such assumption, standard in Bayesian Knowledge Tracing, is that forgetting is negligible, i.e., $forget = 0$. This assumption may be realistic over short time intervals but not in longitudinal studies.

The further from the historical trajectory that we apply the model to explore alternative sequences of tutor decisions, the less we can trust it. Due to the nature of the historical data, estimates of the probability of the student knowing a skill are implicitly conditioned on the student's position in RoboTutor's sequence of activities. Consequently, extrapolating these estimates to states far from the historical trajectory is unreliable.

We tried evaluating policies by their incremental effect – at each point in the history, what action would the policy have chosen? By summing the impact of these individual changes, we hoped to measure the overall impact of the policy more accurately than by following the simulation wherever it led. Unfortunately, this approach turned out to inflate the expected impact by crediting the same beneficial action at multiple simulated steps, despite the fact that in reality it could be performed only once.

Our model ignores item-level effects: it treats all items in an activity as practicing the same skill. This assumption may be plausible for an activity selection policy, but training an item selection policy would require modeling item-level knowledge and learning.

The student and tutor models make some simplifying assumptions. We assume that the student will complete every presented item, and will always choose the first of the two activities offered as choices, rather than the second activity or repeating the previous activity. Rather than try to model students' choices among these options, and their decisions about whether and when to exit an activity, we simply let the RL agent choose which activity to do next after each attempt. The resulting behavior likely overestimates learning compared to real student behavior. Future work could try to avoid these assumptions by training models of student persistence and preference on historical data, but these motivational aspects of student behavior are less understood

than the cognitive aspects modeled by knowledge tracing.

Our HOT-DINA model uses parameters per skill, leaving no differentiation between two activities that exercise the same set of skills. Parameterizing the HOT-DINA model per activity would differentiate between same-skill activities but at the cost of greater computational expense. Future work should explore this option.

Is PPO always best? While our work speaks strongly in favour of PPO and A2C induced policies, we studied deep RL on only one ITS. Future work should study deep RL on diverse tutors to determine if PPO always discovers the most useful policies.

This study used data from only 8 children from one village, out of the hundreds of children who used RoboTutor in 28 villages for the Global Learning XPRIZE's 15-month field test. A future replication study could apply the same approach to historical data from the 27 other villages. More interestingly, testing the learned policy *in vivo* would involve modifying it to stay longer in each activity and devising a method to evaluate its impact, perhaps by comparing the same student's progress under the old and new policies.

## 7. ACKNOWLEDGMENTS

We thank the reviewers of previous versions of this paper for their insightful comments, the chairs of two EDM2021 Workshops for agreeing to transfer this paper to the one it fit better, the dedicated team that continues to develop RoboTutor and analyze its log data, and the XPRIZE Foundation for the opportunity to do so.

## 8. REFERENCES

[1] C. Andrieu, N. de Freitas, A. Doucet, and M. Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50:5–43, 2003.

[2] J. Beck, B. P. Woolf, and C. R. Beal. Advisor: A machine learning architecture for intelligent tutor construction. *In Proceedings of the Seventeenth National Conference on Artificial Intelligence AAAI Press*, pages 552–557, 2000.

[3] O. Bulut. Applying item response theory models to entrance examination for graduate studies: Practical issues and insights. *Eğitimde ve Psikolojide Ölçme ve Değerlendirme Dergisi*, 6, 12 2015.

[4] M. Chi, K. VanLehn, J. Litman, and P. W. Jordan. Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. 2011.

[5] A. Corbett and J. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4:253–278, 1995.

[6] Y. B. David, A. Segal, and Y. K. Gal. Sequencing educational content in classrooms using bayesian knowledge tracing. *Sixth International Conference on Learning Analytics  Knowledge, ACM*, pages 354–363, 2016.

[7] S. Doroudi, V. Aleven, and E. Brunskill. Where's the reward? *Int J Artif Intell Educ*, 2019.

[8] R. K. Hambleton, H. Swaminathan, and H. J. Rogers. Fundamentals of item response theory. *Measurement Methods for the Social Science. Newbury Park, CA: Sage Press*, 1991.

[9] R. A. Howard. Machine-aided learning. *High speed computer system research: quarterly progress report*, 9:19–20, 1960.

[10] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of machine learning research*, pages 1107–1149, December 2003.

[11] D. Lunn, D. Spiegelhalter, A. Thomas, and N. Best. The bugs project: Evolution, critique and future directions. *Statistics in Medicine*, 28:3049–3067, 2009.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglou. Asynchronous methods for deep reinforcement learning. *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 48:1928–1937, 2016.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *Neural Information Processing Systems Deep Learning Workshop*, 2013.

[14] F. E. Ritter, J. Nerb, E. Lehtinen, and T. M. O'Shea. In order to learn: How the sequence of topics influences learning. *Oxford University Press*, August 2007.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[16] S. Shen, M. S. Ausin, B. Mostafavi, and M. Chi. Improving learning  reducing time: A constrained action-based reinforcement learning approach. *In Proceedings of the Conference on User Modeling Adaptation and Personalization, ACM*, 2018.

[17] S. Shen and M. Chi. Reinforcement learning: the sooner the better, or the later the better? 12 2015.

[18] R. Sutton and A. Barto. An introduction to reinforcement learning. *MIT Press*, 1998.

[19] J. Tenenbaum. Bayesian modeling of human concept learning. *Neural Information Processing Systems*, 1999.

[20] L. R. Welch. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, pages 10–13, 2003.

[21] J. Whitehill and J. Movellan. Approximately optimal teaching of approximately optimal learners. *IEEE Transactions on Learning Technologies*, 2017.

[22] R. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[23] Y. Xu and J. Mostow. A unified 5-dimensional framework for student models. *EDM Workshop on Approaching Twenty Years of Knowledge Tracing*, 2014.

[24] M. V. Yudelson, K. R. Koedinger, and G. J. Gordon. Individualized bayesian knowledge tracing models. *International Conference on Artificial Intelligence in Education*, 2013.

[25] G. Zhou, J. Wang, C. F. Lynch, and M. Chi. Towards closing the loop: Bridging machine induced

pedagogical policies to learning theories. *In Proceedings of the 10th International Conference on Educational Data Mining*, pages 112–119, 2017.