A Pre-Trained Graph-Based Model for Adaptive Sequencing of Educational Documents

Jean Vassoyan[1,2]   **Anan Schütt[3]**   **Jill-Jênn Vie[4]**   **Arun-Balajiee Lekshmi-Narayanan[5]**
Elisabeth André[3]   **Nicolas Vayatis[1]**

[1]Université Paris-Saclay, CNRS, ENS Paris-Saclay, Centre Borelli, France
[2]onepoint, France   [3]University of Augsburg, Germany   [4]Soda Team, Inria Saclay, France
[5]University of Pittsburgh, USA

{jean.vassoyan, nicolas.vayatis}@ens-paris-saclay.fr
{anan.schuett, elisabeth.andre}@uni-a.de   arl122@pitt.edu
jill-jenn.vie@inria.fr

Abstract

Massive Open Online Courses (MOOCs) have greatly contributed to making education more accessible. However, many MOOCs maintain a rigid, one-size-fits-all structure that fails to address the diverse needs and backgrounds of individual learners. Learning path personalization aims to address this limitation, by tailoring sequences of educational content to optimize individual student learning outcomes. Existing approaches, however, often require either massive student interaction data or extensive expert annotation, limiting their broad application. In this study, we introduce a novel data-efficient framework for learning path personalization that operates without expert annotation. Our method employs a flexible recommender system pre-trained with reinforcement learning on a dataset of raw course materials. Through experiments on semi-synthetic data, we show that this pre-training stage substantially improves data-efficiency in a range of adaptive learning scenarios featuring new educational materials. This opens up new perspectives for the design of foundation models for adaptive learning.

1 Introduction

One-on-one tutoring has been shown to yield higher learning gains than one-to-many teaching [Bloom, 1984], encouraging the development of *adaptive learning* as a research area, with a wide range of open problems. Learning path personalization is one of them [Brusilovsky and Millán, 2007]: given an e-learning platform with a corpus of educational documents, how can we recommend a sequence of these documents to any student in order to maximize individual learning gains? Personalization is common in computerized adaptive testing, where the questions are chosen on-the-fly based on the performance of the examinee. Knowledge tracing, the task of modeling the acquisition of knowledge, has been done either explicitly using graphical models [Corbett and Anderson, 1994, Doignon and Falmagne, 1985, Leighton et al., 2004], or implicitly using neural networks [Piech et al., 2015, Ghosh et al., 2020].

The problem of learning path personalization can be formulated as a Markov decision process (MDP), where each episode is a learning session with a student, each action is a recommendation (of document) and the reward signal is the learning gains of the student. Consequently, many attempts have been made to tackle this problem with reinforcement learning (RL) algorithms [Clement et al., 2015, Bassen et al., 2020, Shabana et al., 2022]. However, collecting human learning data is expensive, making these approaches very sensitive to the well-known problem of sample efficiency. Therefore,
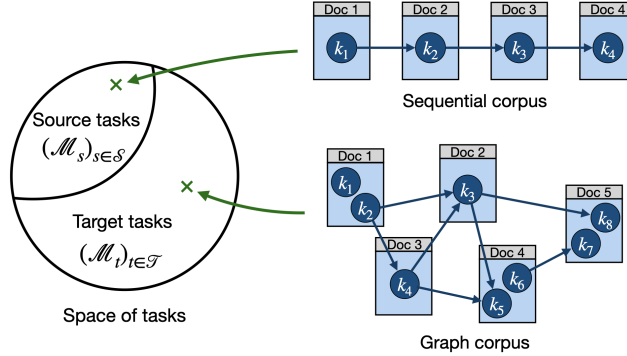
Figure 1: Illustration of the partition of the domain: *sequential* corpora are designed to be followed in one single way whereas *graph* corpora can be navigated along a variety of paths. This dichotomy can be used to pre-train a recommender system on readily available source tasks $(\mathcal{M}_s)_{s \in \mathcal{S}}$ and fine-tune it on a target task $\mathcal{M}_t$.

standard deep RL approaches are usually impractical, as one can hardly afford to make a model interact with thousands of humans.

This falls under a common issue in applied RL where the environment is hardly accessible for training. One prevalent approach to addressing this issue is to train the RL agent in a simulated environment close enough to "real-world" conditions [Towers et al., 2023, Tassa et al., 2018]. However, simulating student behavior generally requires extensive annotation of educational activities with knowledge components and prerequisite relationships [Corbett and Anderson, 1994, Thaker et al., 2020]. These are costly to extract and rarely available in adaptive learning platforms. Another common approach is offline RL [Lange et al., 2012, Levine et al., 2020], which allows training an RL agent on real-world educational data, alleviating the dependence on a specific student simulator. However, the lack of widely available data in adaptive learning environments complicates this approach. Another solution can be found in transfer learning: if the target environment is not easily accessible, one can leverage information from similar - yet more accessible - environments to pre-train the agent. The rich literature on transfer learning in deep RL has been covered by Zhu et al. [2023] and numerous works have demonstrated the benefit of transferring representations from source to target domains [Rusu et al., 2016, Devin et al., 2017, Zhang et al., 2018]. In this paper, we propose to apply such transfer learning strategy to the problem of learning path personalization. Our approach relies on a partition of the domain between two sets of tasks: the *source* and *target* tasks (see Figure 1). The source tasks, referred to as *sequential corpora*, are corpora designed for sequential navigation, as is commonly the case in MOOCs. These are non-adaptive but represent massive amounts of data. The target tasks, referred to as *graph corpora*, denote corpora that can be browsed in multiple ways: these are more suitable for adaptive learning but represent a smaller amount of data.

Our contribution is twofold. We first set up a mathematical framework for modeling the behavior of a student population interacting with a distribution of educational corpora. This framework enables to translate the dichotomy between *sequence* and *graph* corpora into a partition of the domain between *source* and *target* tasks. Then we show experimentally that a recommender system pretrained on a set of educational corpora (source tasks) can transfer its knowledge to a brand new adaptive learning corpus (target task) and benefit from a considerable increase in sample efficiency, especially in the small data regime. To the best of our knowledge, this is the first pre-trained recommender system for learning path personalization. [1]

2   Formalization of the problem

Given a set of educational corpora $\mathscr{C}$ and a student population $\mathcal{P}$, we formalize the problem of learning path personalization as a set of POMDPs. Each episode starts by sampling a corpus $\mathcal{C}$ from $\mathscr{C}$ and a student u from $\mathcal{P}$. The task consists in recommending a sequence of documents from $\mathcal{C}$

---

[1]Our code and experiments are available at:
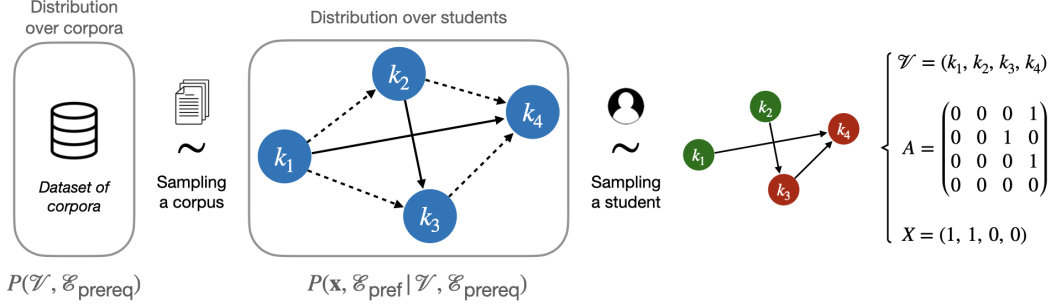https://github.com/jvasso/pretrain-rl-adaptive-learning.

Figure 2: Illustration of the sampling process of a student-corpus pair. A is the adjacency matrix corresponding to the set of edges $\mathcal{E} = \mathcal{E}_{\text{prereq}} \cup \mathcal{E}_{\text{pref}}$.

to maximize the learning gains of student u. We first model the joint distribution over $(u, \mathcal{C})$ as a random graph, then we show how it can be used to formalize the problem as a collection of POMDPs that can be partitioned into *source* and *target* tasks.

2.1 Student and corpus distributions

In the literature on adaptive learning, a corpus $\mathcal{C}$ of educational materials is usually characterized by three elements [Thaker et al., 2020, Leighton et al., 2004]:

- a set of documents $\mathcal{D} = \{d_1, d_2, \dots\}$;
- a set of knowledge components $\mathcal{V} = \{k_1, k_2, \dots\}$; a knowledge component (KC) is a small unit of knowledge taught by a document ; we denote $d_\rightarrow \subset \mathcal{V}$ the set of KCs taught by document d;
- a set of prerequisite relationships $\mathcal{E}_{\text{prereq}} \subset \mathcal{V}^2$ between these KCs ; $(k_i \rightarrow k_j) \in \mathcal{E}_{\text{prereq}}$ means that k_i is a prerequisite for k_j.

On the other hand, the behavior of a student u interacting with $\mathcal{C}$ is determined by these main components:

- his prior knowledge $\mathbf{x}^{(u)} \in \{0, 1\}^{|\mathcal{V}|}$: this is a binary vector stating which KCs were already known by the student prior to the learning session;
- his learning preferences $\mathcal{E}_{\text{pref}}^{(u)} \subset \mathcal{V}^2$: these are similar to the prerequisite relationships $\mathcal{E}_{\text{prereq}}$ but depend on the student[2]; we denote $\mathcal{E}^{(u)} = \mathcal{E}_{\text{prereq}} \cup \mathcal{E}_{\text{pref}}^{(u)}$; we also denote $d_\leftarrow^{(u)} \subset \mathcal{V}$ the requirements of document d for student u: $d_\leftarrow^{(u)} = \bigcup_{k \in d_\rightarrow} \{k' | (k' \rightarrow k) \in \mathcal{E}^{(u)}\}$;
- his ability to demonstrate his knowledge when the opportunity arises; this is typically modeled as a function that maps a knowledge state s and a document d to an observation o of the student's knowledge , through a likelihood function: $\mathcal{Z}(s, d, o) = \mathbb{P}(\mathbf{o}_t = o \mid \mathbf{s}_t = s, \mathbf{d}_t = d)$; we assume that $\mathcal{Z}$ is identical for all students.

We drop superscripts (u) in the following. All previously defined variables play a role in the sampling of a student-corpus pair $(u, \mathcal{C})$. However, in our probabilistic modeling, we only write the key variables $(\mathcal{V}, \mathcal{E}, \mathbf{x})$, as they are the only sources of variability among students. Consequently, a student-corpus pair $(u, \mathcal{C})$ can be modeled as a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{x})$ and the joint distribution over $(u, \mathcal{C})$ can be expressed as a random graph $\mathcal{G} \sim P(\mathcal{V}, \mathcal{E}, \mathbf{x})$. Note that $\mathcal{E}$ and $\mathbf{x}$ are not independent since any knowledge state has to be consistent with the requirements. In practice, we carry out the sampling process in two steps: sampling a corpus, then sampling a student (see Figure 2).

$$P(u, \mathcal{C}) = P(u|\mathcal{C})P(\mathcal{C}) = P(\mathcal{E}_{\text{pref}}, \mathbf{x}|\mathcal{V}, \mathcal{E}_{\text{prereq}})P(\mathcal{V}, \mathcal{E}_{\text{prereq}})$$

---

[2]this choice of modeling learning preferences as an additional set of prerequisite relationships is a particularity of our modeling. It is further motivated in Appendix A.1.

3

2.2 Definition of the domain

We now define the domain, i.e. the set of tasks, as a collection of POMDPs. Given a corpus $\mathcal{C} = (\mathcal{V}, \mathcal{E}_{\text{prereq}})$, a recommendation task is a POMDP $\mathcal{M}(\mathcal{C}) = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{Z})$. $\mathcal{S} \subset \{0,1\}^{|\mathcal{V}|} \times \mathcal{V}^2$ is the state space, which means that each student state $\mathbf{s}_t := (\mathbf{x}_t, \mathcal{E}_{\text{pref}})$ is defined as a combination of the student's knowledge and learning preferences (we assume $\mathcal{E}_{\text{pref}}$ remains constant throughout the episode). $\mathbf{a}_t \in \mathcal{A}$ is the document recommended at step t (also denoted $\mathbf{d}_t$). $\mathbf{o}_t \in \mathcal{O}$ is an observation of the student's knowledge at time t: in our setting, $\mathbf{o}_t$ is a feedback signal qualifying the student's understanding of $\mathbf{d}_t$. This feedback can take 3 possible values: $\mathbf{o}_t = f_<$ if the student did not understand the document (because he didn't have the prerequisites) ; $\mathbf{o}_t = f_\circ$ if he understood the document and learned something new ; $\mathbf{o}_t = f_>$ if he understood the document but did not learn anything new (he already knew all the KCs taught by the document).

This is formalized by the observation function:

$$\mathcal{Z}(s, d, o) = P(\mathbf{o}_t = o \mid \mathbf{s}_t = s, \mathbf{a}_t = d) = \begin{cases} 1 - m_s(d_\leftarrow), & \text{if } o = f_< \text{ (too hard)} \\ m_s(d_\rightarrow), & \text{if } o = f_> \text{ (too easy)} \\ m_s(d_\leftarrow)\big(1 - m_s(d_\rightarrow)\big), & \text{if } o = f_\circ \text{ (right level)} \end{cases} \tag{1}$$

where $s = (\mathbf{x}, \mathcal{E}_{\text{pref}})$ is the student state and $m_s(\cdot)$ is a function that states whether a student in state s masters a given set of KCs: $m_s(\mathbf{v}) = \bigcap_{k_j \in \mathbf{v}} \{x_j = 1\}$. Note that $m_s(d_\leftarrow)$ and $1 - m_s(d_\rightarrow)$ cannot simultaneously be zero, as $d_\leftarrow$ is always a prerequisite of $d_\rightarrow$, making the likelihood function valid.

The transition probability function $\mathcal{T}(s, d, s')$, unknown to the recommender system, simply states that a student returning feedback $f_\circ$ on document d will learn all the knowledge components taught by d. Otherwise the student will remain in the same knowledge state. More formally:

$$\mathcal{T}(s, d, s') = \mathbb{P}(\mathbf{s}_{t+1} = s' \mid \mathbf{s}_t = s, \mathbf{a}_t = d) = \prod_{k_i \in \mathcal{V}} \tau(s, d, x_i') \tag{2}$$

where $s = (\mathbf{x}, \mathcal{E}_{\text{pref}})$, $s' = (\mathbf{x}', \mathcal{E}_{\text{pref}})$ and $\forall i \in \{1, \ldots, |\mathcal{V}|\}$:

$$\tau(s, d, x_i') = \begin{cases} \mathbb{1}_{\{k_i \in d_\rightarrow\}}(1 - m_s(d_\leftarrow)) + \mathbb{1}_{\{k_i \notin d_\rightarrow\}} & \text{if } x_i = x_i' \\ \mathbb{1}_{\{(k_i \in d_\rightarrow) \wedge (x_i' = 1)\}} m_s(d_\leftarrow) & \text{if } x_i = 1 - x_i' \end{cases} \tag{3}$$

A derivation of this formula is provided in Appendix A.2. The reward function $\mathcal{R}(\mathbf{s}_t, \mathbf{s}_{t+1})$, is the learning gains of the student from steps t to step $t + 1$, i.e. the number of newly acquired KCs:

$$\mathcal{R}(\mathbf{s}_t, \mathbf{s}_{t+1}) = |\mathbf{x}_{t+1} - \mathbf{x}_t| \tag{4}$$

where $\mathbf{s}_{t+1} = (\mathbf{x}_{t+1}, \mathcal{E}_{\text{pref}})$, $\mathbf{s}_t = (\mathbf{x}_t, \mathcal{E}_{\text{pref}})$, and $|\cdot|$ is the L^1 norm. As in most reinforcement learning problems, the goal of the above POMDP $\mathcal{M}(\mathcal{C})$ is to find a policy $\pi : \mathcal{O} \to \mathcal{A}$ that maximizes the expected return over each episode: $\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t \mathcal{R}(\mathbf{s}_t, \mathbf{s}_{t+1}) \right]$.

2.3 Partition of the domain

We split the space of corpora into two subspaces: *sequential* corpora and *graph* corpora (see Figure 1). On the one hand, *sequential* corpora relate to "standard" courses, designed to be fully completed, following an identical path for each student. We argue that the set of sequential corpora is a great source of pre-training data for three reasons. First, they represent a huge amount of data (most e-learning curricula are designed that way). Second, since they are designed to be followed sequentially, we consider that each corpus can be reasonably approximated with a chain of KCs where each document teaches one KC (this is reasonable if the documents are small enough): for $\mathcal{C} = (\mathcal{V}, \mathcal{E}_{\text{prereq}})$ with $\mathcal{V} = \{k_1, \ldots, k_n\}$, we have $\mathcal{E}_{\text{prereq}} = \{(k_i \to k_{i+1})\}_{1 \leq i < n}$ and $d_{i\rightarrow} = \{k_i\}$ for all $i = 1, \ldots, n$. This allows to bypass the extensive tagging process usually done by experts. Third, even though it does not leave much room for adaptivity (we have necessarily $\mathcal{E}_{\text{pref}} = \varnothing$), the structure of a sequential course, designed by an expert of the domain, is a great source of information on the relationship among technical concepts. Consequently, we refer to the set of sequential corpora as the set of source tasks, denoted $\mathcal{M}_s$.

On the other hand, *graph* corpora are much more flexible: some knowledge components can be reached through multiple paths, depending on the student's background and learning preferences. In

this case, $\mathcal{C}$ is a non-sequential graph, and $\mathcal{E}_{\text{pref}}$ is not empty. These corpora are much more suited for adaptive learning and, therefore, correspond to the real purpose of our recommendation engine. We call them target tasks and denote $\boldsymbol{\mathcal{M}}_t$ the corresponding set.

Following the formalism of Zhu et al. [2023], we formulate our problem as follows: given a target task $\mathcal{M}_t \in \boldsymbol{\mathcal{M}}_t$, we aim to learn an optimal policy π^* for $\mathcal{M}_t$, by leveraging exterior information from $\boldsymbol{\mathcal{M}}_s$ as well as interior information from $\mathcal{M}_t$.

3 Recommender system: encoder and policy

In this section, we present the architecture of our recommender system. A frequently used approach to solve POMDPs is to leverage information from past observations in order to build an estimate of the current state, i.e to find a function $\phi\colon \mathbf{o}_1 \ldots \mathbf{o}_t \mapsto \mathbf{z}_t$ such that $\mathbf{z}_t$ is a good representation of $\mathbf{s}_t$ [Moerland et al., 2023]. Then this estimation is used to take the next action with a function $\psi\colon \mathbf{z}_t \mapsto \mathbf{a}_t$. ϕ acts as an encoder and ψ acts as a policy. We define our recommender system as a composition of the two functions: $\mathcal{F} = \psi \circ \phi$.

In educational contexts, $\mathbf{z}_t$ is often taken as an estimate of the student's knowledge – usually referred to as the *knowledge state* [Bassen et al., 2020, Reddy et al., 2017]. This knowledge state is traditionally built upon the space of KCs. However, these KCs are difficult to infer in practice. Therefore, following the idea from Vassoyan et al. [2023], we have built our estimate upon a space of *concepts* (or "keywords"). In this context, a "keyword" is a word or group of words that refers to a technical concept closely related to the academic topic of the corpus. These are much easier to extract than KCs, and therefore better suited for our generalization purposes (we detail in Appendix F how we automatically extract keywords from corpora using an LLM). Consequently, we model the student's knowledge state as a collection of keyword vectors $\mathbf{z}_t = (\mathbf{h}_{w_1}, \mathbf{h}_{w_2}, \dots)$ intended to capture his knowledge of each concept (see Figure 5 in the appendix) [3].

Regarding the architecture of ϕ and ψ, note that our framework requires a flexible model that must be able to run on multiple corpora (with different sizes). Graph neural networks (GNN) have long been identified as an efficient way to make recommendations on flexible data structures: indeed, their number of parameters does not depend on the size of the support [Wang et al., 2019]. Therefore we define ϕ and ψ as GNNs operating on a bipartite graph of documents and keywords $\mathcal{B} = \big((\mathcal{V}_{\mathcal{D}}, \mathcal{V}_{\mathcal{W}}), \mathcal{E}_{\mathcal{B}}, (E, F)\big)$, where $\mathcal{V}_{\mathcal{D}}$ is the set of documents, $\mathcal{V}_{\mathcal{W}}$ is the set of keywords and $\mathcal{E}_{\mathcal{B}}$ the set of edges with $(d, w) \in \mathcal{E}_{\mathcal{B}}$ if the document d contains the keyword w. Regarding the node features, we have used pre-trained word embeddings E for the keyword nodes and the student's past feedback F for the document nodes[4]. Eventually, for the layers of the GNNs, we have used the graph transformer operator from Shi et al. [2021]:

$$\mathbf{h}_i^{(l+1)} = \mathbf{W}_1 \mathbf{h}_i^{(l)} + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \mathbf{W}_2 \mathbf{h}_j^{(l)} \quad \text{with} \quad \alpha_{i,j} = \text{softmax}\Big(\frac{\big(\mathbf{W}_3 \mathbf{h}_i^{(l)}\big)^{\top} \big(\mathbf{W}_4 \mathbf{h}_j^{(l)}\big)}{\sqrt{n}} \Big) \quad (5)$$

where $\mathbf{h}_i^{(l)} \in \mathbb{R}^n$ is the embedding of node i at layer l ($h_i^{(0)}$ are the features of the nodes), $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4 \in \mathbb{R}^{n \times n}$ are the weight matrices and $\alpha_{i,j} \in \mathbb{R}$ are the attention coefficients, computed via multi-head dot product attention. This model is similar to the one provided by [Vassoyan et al., 2023].

Note that in the end, there are two graphs: the environment graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{x})$ that determines the students behaviour (unknown to the recommender system) and the bipartite graph $\mathcal{B} = \big((\mathcal{V}_{\mathcal{D}}, \mathcal{V}_{\mathcal{W}}), \mathcal{E}_{\mathcal{B}}, (E, F)\big)$ that is passed as an input to the recommender system. A global view of the recommendation pipeline is provided in Figure 3. More details about the architecture of the model and its hyperparameters can be found in Appendix B.

---

[3]Note that even though our experimental pipeline uses KCs to simulate students and properly evaluate the models, in practice our recommender system does not rely on them and makes recommendations solely based on keyword embeddings and human interactions.

[4]Actually for the document nodes, we have used a combination of keyword embeddings and student's past feedback: more details are provided in Appendix B.
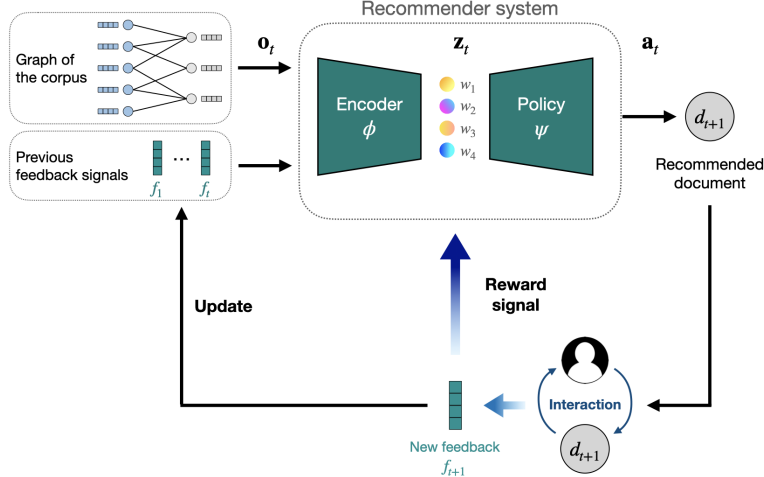
Figure 3: Overview of the recommendation pipeline on one student

4 Pre-training on source tasks

In this section, we present our approach for pre-training our recommender system on sequential corpora.

4.1 Data distribution

Corpus For the corpus distribution, we have used 14 real-world **sequential** corpora scraped from 3 popular e-learning platforms. Each corpus contains a sequence of videos that are meant to be viewed one after another, as they build up from each other. These corpora address educational topics related to machine learning, statistics, and computer science. We automatically extracted keywords from the transcriptions of these videos using an LLM, as described in Appendix F. For the keyword features, we have used pre-trained embeddings from Wikipedia2Vec [Yamada et al., 2020].

Population For a sequential corpus, $\mathcal{E}_{\text{pref}} = \varnothing$. Therefore the only degree of freedom is the distribution over prior knowledge $P(\mathbf{x}|\mathcal{V}, \mathcal{E}_{\text{prereq}})$ with $\mathcal{E}_{\text{prereq}} = k_1 \rightarrow k_2 \cdots \rightarrow k_N$. For this pre-training task, we assumed a population with zero prior knowledge for simplicity. This means that for each student, $\mathbf{x}_0 = (0 \ldots 0)$ at the beginning of the learning session.

4.2 Pre-training

The pre-training was carried out in two stages. In the first stage, we trained the model to match the predictions of an oracle, in a supervised learning way. Our oracle is simply an algorithm that has access to the environment graph and therefore knows exactly which document to recommend in each situation. Note that this first stage is quite similar to the way LLMs are pre-trained: instead of predicting the next token, we predict the next document in a predefined sequence. In the second stage, a reinforcement learning training was performed with the REINFORCE algorithm [Sutton et al., 1999]. This stage is crucial as, with a discount factor greater than zero, the model can learn to make "useful" mistakes —that is, recommendations that do not facilitate student learning but provide valuable information about his current knowledge. The choice of not using any algorithm involving an approximate value function, although these are known to be more stable [Mnih et al., 2013, Haarnoja et al., 2018, Mnih et al., 2016], was motivated by the difficulty for GNNs to reason at the scale of the whole graph (which is required for a good estimate of the state value). We directly trained our RL agent on the collection of 14 POMDPs induced by our sequential corpora. The hyperparameters we have used for this pre-training stage as well as the corresponding learning curve are provided in Figure 6 of the Appendix. We have trained the RL agent on approximately 25k steps. The maximum length of each episode was set according to the number of documents in the current corpus. Other pre-training strategies have also been tested, but with weaker results. Results and discussion on these are provided in Appendix E.

5 Fine-tuning on target task

5.1 Data distribution

Corpus To evaluate our recommender system in a complex adaptive learning scenario, we have designed a corpus of 22 written documents teaching machine learning basics. We have conceived it in a way that the dependencies between knowledge components are easy to establish manually. The resulting graph is a grid of KCs with 3 rows and 11 columns, depecting prerequisite relationships among all documents. We have also included an additional "background" KC that isn't taught by any document but conditions the access to some of them, thus bringing greater diversity to the student population. More details about the design of this corpus along with a graphical representation are provided in Appendix D.1.

Population In our framework, choosing a student distribution is equivalent to defining $P(\mathcal{E}_{\text{pref}}|\mathcal{C})$ and $P(\mathbf{x}|\mathcal{E}_{\text{pref}}, \mathcal{C})$. Regarding $P(\mathcal{E}_{\text{pref}}|\mathcal{C})$, we considered that in the specific case of our corpus, it made sense to model learning preferences as a set of extra "vertical" dependencies (edges) between knowledge components (see Figure 7 of the Appendix). Therefore, for each new student, a random set of such edges was generated, following a binomial distribution of parameter $p = 0.3$. So for any $k_{i,j} \in \mathcal{V}, P(k_{i,j} \to k_{i+1,j} \in \mathcal{E}_{\text{pref}}) = 0.3$. An interpretation of this modeling of learning preferences is provided in Appendix D.1. As for the distribution over prior knowledge $P(\mathbf{x}|\mathcal{E}_{\text{pref}}, \mathcal{C})$, we have considered 3 possible scenarios:

Scenario 1 (none): the learners have no prior knowledge, i.e. $\mathbf{x} = (0 \ldots 0)$ for all students;

Scenario 2 (decreasing exponential): the number of KCs known by the students prior to the learning session follows a decreasing exponential distribution; this allows to model a population where most students have little knowledge about the corpus, but not all of them;

Scenario 3 (uniform): the number of KCs known by the students prior to the learning session follows a uniform distribution; this is the most challenging environment as uncertainty is maximal.

5.2 Training

For each sampled student, some paths in the corpus are reachable while some others are not (depending on their prior knowledge and learning preferences). To avoid that the agent always chooses the safest path (instead of adapting to the student's profile), we slightly modified the reward function, in order to give greater rewards to "difficult" paths. Rather than summing the newly acquired knowledge components as depicted in Equation 4, we computed a weighted sum by multiplying each KC by its respective value. More details about the values of the KCs are provided in Appendix D.1.

The fine-tuning was performed on 10 epochs, where 5 new students were sampled on each epoch. Therefore, it was carried out on a total of 50 students. The maximum size of each episode was set to 11 (because of the 11 "major" concepts, cf. Appendix D.1). On each epoch, we tested our model on 20 test episodes ($\sim$ 20 students) and aggregated our results over 30 random seeds. We did not perform any hyperparameter search in the fine-tuning stage so as not to give an advantage to our pre-trained model. Instead, we simply adopted the hyperparameters used by Vassoyan et al. [2023] as they provided good performance for training on a single corpus (reported in Table 3 of the Appendix).

Prior knowledge	Ours	Vassoyan et al.	CMAB	Bassen et al.
None	**24.81 (2.63)**	18.63 (6.55)	18.34 (2.11)	8.02 (1.62)
Decreasing exp.	**22.62 (1.82)**	16.28 (4.51)	11.32 (2.24)	4.64 (1.07)
Uniform	**13.33 (3.40)**	7.51 (3.44)	4.19 (1.68)	2.49 (0.70)

Table 1: Final episodic returns (average and standard deviation) on each task.

5.3 Baselines

Although numerous approaches had been explored in the literature on learning path personalization, for a fair comparison we selected only models that did not use the corpus's prerequisite graph for making predictions. Consequently, we compared our approach against three baselines.
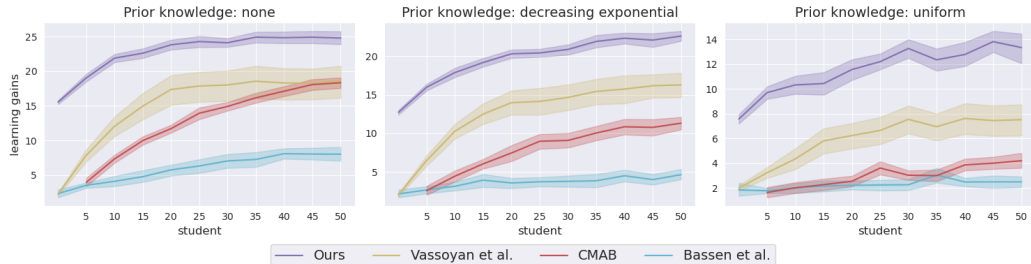
Figure 4: Performance comparison (mean and confidence interval) depending on the pre-training strategy, for multiple prior knowledge distributions in the population. At each epoch, new data are collected from 5 new sessions with simulated students. In total, the training is carried out on a group of 50 students. These results are aggregated over 30 random seeds.

First is the approach from Bassen et al. [2020], which was used to recommend a sequence of educational activities to students. Their model consists of 2 fully-connected neural networks (one for the actor and one for the critic) trained with Proximal Policy Optimization [Schulman et al., 2017]. Since our learning scenario is quite different from theirs, we had to adapt the observation space: in particular, the pre-test scores and the scores on activities were omitted. Instead, we just kept a feature vector stating which feedback was given on each document by the current student. The authors did not provide the hyperparameters they used in their experiment. Therefore, we carried out a hyperparameter search over various learning rates, layer sizes, batch sizes and "repeat per collect" [5] values and kept the best.

Our second baseline was the linear contextual multi-armed bandit (CMAB), with Thompson Sampling as the policy Agrawal and Goyal [2013]. The bandit requires context for each action instead of states, which we provided as the number of times the document was previously recommended to the student, and the number of times it was "understood".

Finally, we compared our approach against the model from Vassoyan et al. [2023], which is very similar to ours, except that it does not feature any type of pre-training.

5.4 Results and discussion

The results of our fine-tuning experiments are presented in Figure 4 and Table 1. The four recommendation engines are compared in each of the three scenarios described in 5.1. The performance of each model is measured in learning gains per student, which, in our setting, is equivalent to the undiscounted episodic return. These learning gains are plotted against the number of students, where each student can be regarded as an episode. The maximum episode length is 11, which puts us in a very low data regime. The margins of error in Figure 4 were estimated using a studentized bootstrap method, with 10,000 resamples and a 95% confidence interval.

The first observation that can be made from these plots is that our pre-trained model outperformed all other approaches in all scenarios. As expected, the performance gap was most pronounced at the beginning of the training sessions, where our model benefited from a strong warm start. Moreover, it maintained its lead throughout the entire training sessions and converged to values well above those of its non-pre-trained counterparts. This suggests that the pre-training process has given our model some notion of dependencies among keyword embeddings, helping it to quickly find good recommendation strategies, even though it had never previously encountered the target corpus.

Surprisingly enough, our model seems to perform just as well in scenarios 2 and 3, despite not having been exposed to them in the pre-training stage. This shows that in the pre-training stage, the model has acquired insights that extend beyond mere knowledge of the population's distribution. In comparison, a simple baseline like CMAB performed quite well when the population had no prior knowledge but struggled more and more as the uncertainty increased. The model from [Bassen et al.,

---

[5]This parameter specifies the number of times the policy is updated for each batch of data collected. For instance, setting it to 2 indicates that the policy will undergo two learning iterations for each batch of collected data.

2020] got the worst performance, despite being the only one that benefited from a hyperparameter search on this corpus. One explanation could be that this model was designed for a real world setting with fewer educational activities and much more students (~ 1000).

Eventually one can note from Table 1 that despite being trained with the REINFORCE algorithm, which, in comparison to PPO, usually suffers from high variance, our model showed a relatively low variance (much lower than its non-pre-trained counterpart).

6   Related work

Previous works have used reinforcement learning to decide on the pedagogical action in computer-based education to increase the student's learning gain. Many works used RL agents with small state and action spaces to accommodate the small amount of data available. For such cases, a tabular RL algorithm was used to train an RL agent. One example of this is the work by Chi et al. [2011], where the RL agent takes a state vector containing e.g. the simplicity of the concept to explain, and decides between hinting and telling the student how to solve an exercise. Similar approaches have been shown to work in practice in specific and tightly constrained settings [Gordon et al., 2016, Park et al., 2019]. Because of the simplicity of the state space, these works are limited to small and fixed, specific sets of interactions.

Another branch of approaches used simulated students to allow for more learning iterations. Among the first was the work by Iglesias et al. [2009], where tabular Q-learning was used to choose the types of learning material for simulated students. Later works have used RL algorithms that can handle more complex state and action spaces [Lan and Baraniuk, 2016, Rafferty et al., 2016], and in particular deep RL [Reddy et al., 2017, Subramanian and Mostow, 2021]. Although simulated students allow RL agents to train for virtually unlimited episodes, and so handle more complex dynamics, there is a strong reliance on the accuracy and realism of the simulated students. We alleviate this by using simulated students for pre-training, then evaluate the potential of the RL agent to generalize to an unseen learning environment.

Closely related to our work was by Bassen et al. [2020], which applied deep RL to recommend different educational materials from a pool of 12. They used data collected from 1,830 participants to train the agent. They used one-hot encoding to represent the materials, meaning that the agent needs to be retrained to work with another set of materials. Their approach also required a large number of real student data because of the sparse reward, coming only once at the end of an episode, showing the importance of sample efficiency in real-world applications. We improve upon this by using meaningful embeddings to represent the materials to recommend.

Designing simulated students requires modeling how knowledge works. A student's knowledge is broken down into Knowledge Components (KCs), which are small units of knowledge that a student can learn and hold (e.g. a fact or a skill). Based on this, different exercises or content can be associated with different KCs, creating a binary matrix called the q-matrix [Barnes, 2005], allowing interactions with an exercise to act as evidence for the associated KCs. Building up further, the theory of knowledge space and the attribute hierarchy method both suggest an order of prerequisites between these KCs, which can be represented as a graph [Doignon and Falmagne, 1985, Leighton et al., 2004].

7   Conclusion

Our study shows that when dealing with an adaptive learning scenario, one efficient strategy to improve sample-efficiency is to gather a set of closely related sequence corpora and pre-train a recommender system on them. This pre-training procedure does not involve any annotator to tag documents nor real students to interact with them, but requires a flexible enough recommender system. This opens new avenues for applications of deep RL algorithms to personalized learning environments, and paves the way for the developement of scalable and reusable models for adaptive learning. In the future, this method could be tested on a wider range of adaptive learning scenarios as well as on real-world student populations.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International conference on machine learning*, pages 127–135. PMLR, 2013.

Tiffany Barnes. The q-matrix method: Mining student response data for knowledge. In *American association for artificial intelligence 2005 educational data mining workshop*, pages 1–8. AAAI Press, Pittsburgh, PA, USA, 2005.

Jonathan Bassen, Bharathan Balaji, Michael Schaarschmidt, Candace Thille, Jay Painter, Dawn Zimmaro, Alex Games, Ethan Fast, and John C Mitchell. Reinforcement learning for the adaptive scheduling of educational activities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

Benjamin S Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher*, 13(6):4–16, 1984.

Peter Brusilovsky and Eva Millán. User models for adaptive hypermedia and adaptive educational systems. In *The adaptive web: methods and strategies of web personalization*, pages 3–53. Springer, 2007.

Harrison Chase. LangChain, October 2022. URL https://github.com/langchain-ai/langchain.

Min Chi, Kurt VanLehn, Diane Litman, and Pamela Jordan. Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. *User Modeling and User-Adapted Interaction*, 21:137–180, 2011.

Benjamin Clement, Didier Roy, Pierre-Yves Oudeyer, and Manuel Lopes. Multi-armed bandits for intelligent tutoring systems. *Journal of Educational Data Mining*, 7(2), 2015.

Albert T Corbett and John R Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4:253–278, 1994.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2169–2176. IEEE, 2017.

Jean-Paul Doignon and Jean-Claude Falmagne. Spaces for the assessment of knowledge. *International journal of man-machine studies*, 23(2):175–196, 1985.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Aritra Ghosh, Neil Heffernan, and Andrew S Lan. Context-aware attentive knowledge tracing. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2330–2339, 2020.

Goren Gordon, Samuel Spaulding, Jacqueline Kory Westlund, Jin Joo Lee, Luke Plummer, Marayna Martinez, Madhurima Das, and Cynthia Breazeal. Affective personalization of a social robot tutor for children's second language skills. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Ana Iglesias, Paloma Martínez, Ricardo Aler, and Fernando Fernández. Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. *Applied Intelligence*, 31:89–106, 2009.

Andrew S Lan and Richard G Baraniuk. A contextual bandits framework for personalized learning action selection. In *EDM*, pages 424–429, 2016.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning: State-of-the-art*, pages 45–73. Springer, 2012.

Jacqueline P Leighton, Mark J Gierl, and Stephen M Hunka. The attribute hierarchy method for cognitive assessment: A variation on tatsuoka's rule-space approach. *Journal of educational measurement*, 41(3):205–237, 2004.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

Hae Won Park, Ishaan Grover, Samuel Spaulding, Louis Gomez, and Cynthia Breazeal. A model-free affective reinforcement learning approach to personalization of an autonomous social robot companion for early literacy education. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 687–694, 2019.

Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 505–513. Curran Associates, Inc., 2015.

Anna N Rafferty, Emma Brunskill, Thomas L Griffiths, and Patrick Shafto. Faster teaching via pomdp planning. *Cognitive science*, 40(6):1290–1332, 2016.

Siddharth Reddy, Sergey Levine, and Anca Dragan. Accelerating human learning with deep reinforcement learning. In *NIPS workshop: teaching machines, robots, and humans*, 2017.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *NIPS Deep Learning Symposium*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

KM Shabana, Chandrashekar Lakshminarayanan, and Jude K Anil. Curriculumtutor: An adaptive algorithm for mastering a curriculum. In *International Conference on Artificial Intelligence in Education*, pages 319–331. Springer, 2022.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1548–1554. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/214. URL https://doi.org/10.24963/ijcai.2021/214. Main Track.

Jithendaraa Subramanian and Jack Mostow. Deep reinforcement learning to simulate, train, and evaluate instructional sequencing policies. In *Spotlight presentation at Reinforcement Learning for Education workshop at Educational Data Mining 2021 conference*, 2021.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Khushboo Thaker, Lei Zhang, Daqing He, and Peter Brusilovsky. Recommending remedial readings using student knowledge state. *International Educational Data Mining Society*, 2020.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL https://zenodo.org/record/8127025.

Jean Vassoyan, Jill-Jênn Vie, and Pirmin Lemberger. Towards Scalable Adaptive Learning with Graph Neural Networks and Reinforcement Learning. In *Proceedings of the 16th International Conference on Educational Data Mining*, pages 351–361. International Educational Data Mining Society, July 2023. doi: 10.5281/zenodo.8115663. URL https://doi.org/10.5281/zenodo.8115663.

Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022. URL http://jmlr.org/papers/v23/21-1127.html.

Ikuya Yamada, Akari Asai, Jin Sakuma, Hiroyuki Shindo, Hideaki Takeda, Yoshiyasu Takefuji, and Yuji Matsumoto. Wikipedia2Vec: An efficient toolkit for learning and visualizing the embeddings of words and entities from Wikipedia. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 23–30, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.4. URL https://aclanthology.org/2020.emnlp-demos.4.

Amy Zhang, Harsh Satija, and Joelle Pineau. Decoupling dynamics and reward for transfer learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1aoddyvM.

Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

A Details about the environment

A.1 Learning preferences

We argue that learning preferences can be modelled without loss of generality through a set of additional dependencies between knowledge components (if necessary by adding more knowledge components). To illustrate this point, let's consider two students u_1 and u_2 mastering the same set of knowledge components: $\mathbf{x}_{u_1} = \mathbf{x}_{u_2} = \mathbf{x}$. Let's suppose that u_1 and u_2 hold different learning preferences: this means that there exists at least one document d for which u_1 and u_2 will react differently, for example u_1 understands it while u_2 does not. Assuming that d is not definitively out of scope for student u_2, there must exist a minimal set of documents $\mathcal{D}'$ whose reading will enable him to understand document d. Denoting $\mathcal{K} = d_{\rightarrow}$ and $\mathcal{K}' = \cup_{d' \in \mathcal{D}'} d'_{\rightarrow}$, this translates into an extra set of dependencies $\mathcal{K}' \rightarrow \mathcal{K}$ for student u_2.

A.2 Transition probability function

To describe the transition probability of the student's dynamics, we start with the base case, where there is only one KC in the dynamics. We model the student's knowledge state as a binary "known" or "unknown" state on the KC. The student never forgets, so a known KC remains known for the rest of the episode. Learning can occur when a student interacts with a document that teaches a KC that the student previously did not know. If the student knows all the prerequisite KCs required to understand the document, then the student will learn all the KCs taught by that document. Otherwise, the student will not learn any of the KCs taught. As a result, the transition of the student's state is fully deterministic, dependent on the stochastic and unknown prerequisites of the documents. Formally, the transition probability in the case of 1 KC is described as the following: $\tau(\mathbf{x}, d, x'_i) = \mathbb{P}(\mathbf{s}_{t+1}^{(i)} = x'_i \mid \mathbf{s}_t = \mathbf{x}, \ \mathbf{a}_t = d)$:

$$\tau(\mathbf{x}, d, x'_i) = \begin{cases} 1, & \text{if} \quad (x_i = 1, \ x'_i = 1) \\ 0, & \text{if} \quad (x_i = 1, \ x'_i = 0) \\ \mathbb{1}(k_i \in d_{\rightarrow}) \, \sigma \left(\displaystyle\sum_{k_j \in d_{\leftarrow}} \mathbb{1}(x_j = 0) \right) \\ \quad + \mathbb{1}(k_i \notin d_{\rightarrow}), & \text{if} \quad (x_i = 0, \ x'_i = 0) \\ \mathbb{1}(k_i \in d_{\rightarrow}) \displaystyle\prod_{k_j \in d_{\leftarrow}} \mathbb{1}(x_j = 1), & \text{if} \quad (x_i = 0, \ x'_i = 1) \end{cases} \tag{6}$$

Through straightforward manipulations, we can transform the four cases of Equation 6 into Equation 3.

We then extend the student's dynamics to work with multiple KCs, which we build up from the base case. For this, we model the learning of the different KCs independently, meaning that the transition of one KC does not affect how another KC is learned. Using the notations from Section 2, the full transition probability function on multiple KCs is:

$$\mathbb{P}(\mathbf{s}_{t+1} = s' \mid \mathbf{s}_t = s, \ \mathbf{a}_t = d) = \prod_{k_i \in \mathcal{V}} \mathbb{P}(\mathbf{s}_{t+1}^{(i)} = s'_i \mid \mathbf{s}_t = s, \ \mathbf{a}_t = d) \tag{7}$$

A.3 Observation function

Next is the observation function, which describes how a student responds to a recommended document. There are three possible interactions for the student: right level, too easy, or too hard, denoted $f_{\circ}$, $f_{>}$, $f_{<}$ respectively. The student deems the document too easy when the student has already mastered the KCs that the document teaches. In contrast, the student deems the document too hard when the student did not master all the prerequisite KCs of the document at the time. Otherwise, the document is at the right level, making the student master new KC(s). Formally, the observation function $\mathcal{Z}(s, d, o)$ is described as follows, where $s = (\mathbf{x}, \mathcal{E}_{\text{pref}})$:

$$\mathcal{Z}(s, d, o) = \begin{cases} \mathbb{1}(\mathbf{x} \not\supseteq d_{\leftarrow}), & \text{if} \quad (o = f_{<}) \\ \mathbb{1}(d_{\rightarrow} \subseteq \mathbf{x}), & \text{if} \quad (o = f_{>}) \\ \mathbb{1}(\mathbf{x} \supseteq d_{\leftarrow} \wedge x_i \cap d_{\rightarrow} \neq x_i), & \text{if} \quad (o = f_{\circ}) \end{cases} \tag{8}$$
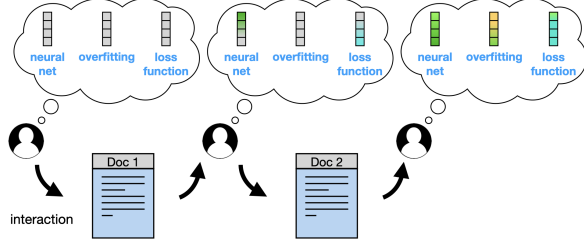
Figure 5: An intuitive view of student's knowledge state model with keyword vectors

B   Architecture of the recommender system

Our recommender system $\mathcal{F} = \psi \circ \phi$ is a composition of two graph neural networks operating on the bipartite graph $\mathcal{B}$ defined in Section 3. GNN layers are implemented with directed edges, successively used from documents to keywords, and from keywords to documents. Moreover, ϕ combines a GNN architecture with a multi-layer perceptron (MLP) that produces embeddings of the feedback signals. These feedback embeddings are then combined with the embeddings of their associated document nodes with a Hadamard product.

Below is the complete architecture of our recommender system:

$$
\left.
\begin{aligned}
H^{(1)} &= \mathsf{Linear}(E) \\
H_W^{(2)} &= \mathsf{TransformerConv}_{\text{doc}\to\text{kw}}(H^{(1)}) \\
H_D^{(2)} &= \mathsf{TransformerConv}_{\text{kw}\to\text{doc}}(H^{(2)}) \\
H_D^{(3)} &= H_D^{(2)} \odot \mathsf{MLP}(F_D) \\
H_W^{(3)} &= \mathsf{TransformerConv}_{\text{doc}\to\text{kw}}(H^{(3)})
\end{aligned}
\right\} \phi \tag{9}
$$

$$
\left.
\begin{aligned}
H_D^{(4)} &= \mathsf{TransformerConv}_{\text{kw}\to\text{doc}}(H^{(3)}) \\
Z_D &= \mathsf{Linear}(H_D^{(4)})
\end{aligned}
\right\} \psi \tag{10}
$$

where $H^{(l)} \in \mathbb{R}^{|\mathcal{V}| \times k}$ is the node embeddings at layer l, with $H_D^{(l)} \in \mathbb{R}^{|\mathcal{V}_D| \times k}$ and $H_W^{(l)} \in \mathbb{R}^{|\mathcal{V}_W| \times k}$ are the restrictions of $H^{(l)}$ to document and keyword nodes respectively. Note that $H_W^{(3)}$ is actually the latent representation $\mathbf{Z}_t$ defined in section 3. $E \in \mathbb{R}^{n \times n}$ is the input node features. F_D is the matrix of past feedback signals (one feedback per document, with a "none" feedback for documents that have not been visited yet). $\mathsf{TransformerConv}(\cdot)$ is the operator formulated in Equation 5, with the subscripts doc $\to$ kw and kw $\to$ doc denoting the directions of edges. $\mathsf{Linear}(\cdot)$ is a linear transformation. $\mathsf{MLP}(\cdot)$ is a two layers perceptron with one hidden layer. $\odot$ is the Hadamard product. An extra linear layer is added after the final $\mathsf{TransformerConv}$ to map document representations to scores (then converted to probability distribution via softmax).

An exhaustive list of the hyperparameters used in our model is provided in Table 2.

Hyperparameter	Value
Hidden dimension	128
Activation function	Exp. Linear Unit (ELU)
Attention type	Additive
Number of attention heads	4
Wikipedia2Vec embedding size	100

Table 2: Hyperparameters of the recommender system

Pre-training on 14 corpora

Hyperparameter	Value
Discount factor	0.7
Learning rate	0.0001
Entropy coefficient	0.01
Batch size	8
Repeat per collect	15
Steps per collect	1024

Figure 6: Learning curve and hyperparameters used for the RL pre-training stage

C   Pre-training on source tasks

The learning curve and hyperparameters we have used for the pre-training on source tasks are provided in Figure 6. The batch size refers to the number of graphs passed as input to the model (hence the small value).

D   Fine-tuning on target task

D.1   Graph corpus

The graph corpus contains a set of written documents on machine learning, which are distinct from the ones in the sequential corpora, albeit they share some keywords. We have designed this corpus in such a way that it is aimed at two types of student: those with Computer Science (CS) background, and those without CS background. Consequently, the corpus teaches 11 major concepts and each concept is addressed by 2 documents, from 2 different angles: the "computer science" angle (often involving linear algebra or IT concepts) and the "non-computer science" angle. To model this, we have adopted a 3-line system of knowledge components, where each major concept is broken down into three knowledge components (one at each level): the middle KC represents the major concept to be taught, the top one represents its "non-CS" angle and the bottom one represents its "CS" angle. Consequently, all the bottom KCs are conditioned by a "background" KC, which states whether the student has a "CS" background or not. Non-CS students can only learn from non-CS documents, as they lack the CS prerequisite. CS students can learn from both types of documents. Prerequisites relationships $\mathcal{E}_{\text{prereq}}$ were modeled as horizontal dependencies in the two bottom levels. As for learning preferences $\mathcal{E}_{\text{pref}}$, these were modeled as "vertical" dependencies. Here is an example to motivate this choice: consider that one major concept of the corpus is the artificial neuron. It can be taught using CS notions (like linear algebra) or without these notions (for instance using a biological analogy). For some CS students, the mathematical explanation may be sufficient to enable them to understand the concept intuitively. But for some others, although they have the necessary CS background, some non-mathematical explanation might help in grasping the concept intuitively. Hence the existence of certain vertical dependencies between the "non-CS" and "CS" levels, which we model as learning preferences. Overall, the graph corpus aims to show that the RL agent can generalize its knowledge from the sequential corpora, and that the RL agent can learn to adapt to different types of students.

The "value" of each KC depends on its position on the grid: the value is 1 for row 1, 2 for row 2 and 3 for row 3 (the most difficult to access are the most valuable).
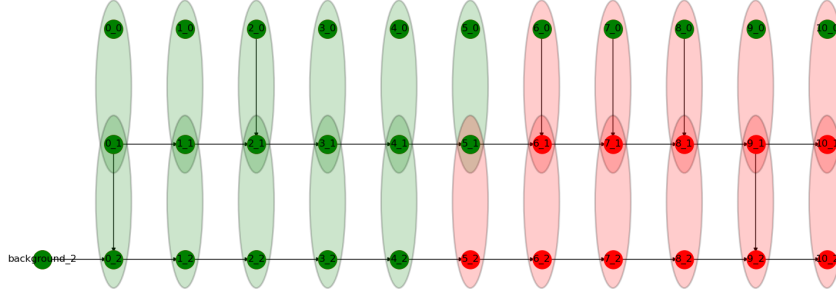
Figure 7: Illustration of the graph corpus we have used for the fine-tuning task. The nodes are the knowledge components and the ellipses that enclose them are the documents. This graph actually illustrates the sampling of a student, as the green color represents the prior knowledge of the student and the vertical edges represent his learning preferences.

D.2 Fine-tuning process

The hyperparameters we have used for the fine-tuning of the model on the target task are listed in Table 3.

Hyperparameter	Value
Learning rate	0.0005
Batch size	16
Repeat per collect	15
Episodes per collect	5
Discount factor	0

Table 3: Hyperparameters used for the RL fine-tuning

E Other pre-training strategies

In this section we provide experimental results for two other pre-training strategies. Both of them are performed on the same source tasks (sequential corpora) we have used in the main experiment. The first strategy is just the same as the one we presented in the core of the paper, without the RL stage. This means the model was solely pre-trained on expert data, allowing to evaluate the importance of the RL stage. The second strategy consists in training the model to predict student's feedback on each next document (instead of choosing next document). This provides a target signal slightly more informative than the sole good/bad recommendation signal we have used for pre-training on expert data. Therefore, this could lead to more accurate internal representations. The learning curves are presented in Figure 8 (our final model is also displayed for comparison). The margins of error were estimated using a studentized bootstrap method, with 10,000 resamples and a 95% confidence interval.
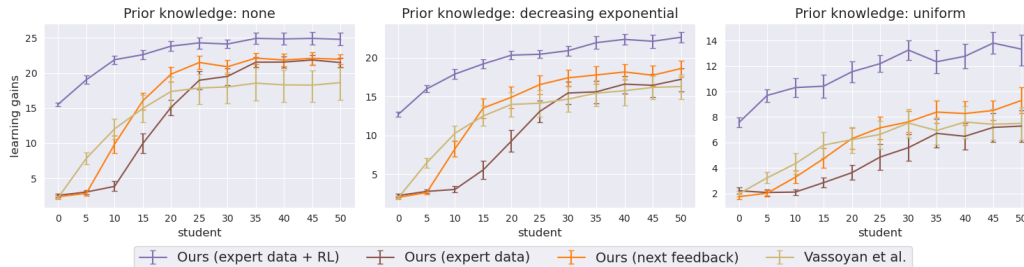

Figure 8: Performance comparison of our model against two other pre-training strategies.

The first takeaway of this experiment is that these two methods, even if they manage to slightly outperform their non-pre-trained counterpart (Vassoyan et al.), perform much worse than our main model. This is especially pronounced at the beginning of the fine-tuning. In the case of the model pre-trained on predicting next feedback, such cold-start is not surprising, as the model is just discovering the recommendation tasks. However, the model pre-trained on expert data was trained precisely on a recommendation task, and its poor performance at the start of fine-tuning is all the more surprising. Although this phenomenon is difficult to interpret at this stage, it does show the importance of RL in pre-training, which alone enables to make good recommendations from the start of fine-tuning.

F Keywords extraction

The main difficulty in our keyword-extraction setting is that we did not extract "regular" keywords but only keywords related to Wikipedia articles. Indeed, we have chosen to use Wikipedia2Vec [Yamada et al., 2020] pre-trained embeddings as features of the keywords, mainly because we consider that embeddings which primarily contain "encyclopedic" information would be more suitable for this task. Since modern large language models usually have a good knowledge of Wikipedia, we simply prompted GPT-4 [Achiam et al., 2023] to achieve this task, using the prompt below (adapted for each document):

System Prompt:

You are a helpful assistant who extract keywords from educational documents. A user will pass an educational document. First, you extract the academic topic of the document. This could be the title of the document in a course sequence. Don't write anything else. Then, you examine every single term in this document and extract them if they refer to a technical concept closely related to the academic topic of the document. Avoid including keywords that are only used in examples and don't have much to do with the subject taught in the document. You directly output a python list of strings (for example: "machine learning", "neural network" etc.), without duplicate. There should never be more than thirty keywords or so, so choose the most important ones. Again, don't write anything else. Eventually, you associate each term with its corresponding Wikipedia page (Wikipedia ID) if it exists. Sometimes this needs to be done cleverly: for example, "sequential data" should be associated with the Wikipedia ID "Sequence". Drop the terms that don't have a Wikipedia page. Directly output a python list of strings of Wikipedia IDs (for example: ["Machine_learning", "Artificial_neural_network", ...]). Again, don't write anything else.

Human Prompt:

Machine Learning allows computers to make decisions without being explicitly programmed to do so. Imagine you are teaching a child to recognize different fruits. You might show the child various pictures of apples, bananas, oranges, and so on. Over time, the child learns to identify these fruits correctly. Machine learning works in a similar way. You feed the computer a lot of data—like pictures of fruits—and the computer learns to recognize patterns within that data to differentiate between them. After sufficient learning, it can correctly identify or predict new, unseen data. For example, if you show it a new picture of an apple, it will be able to identify it as an apple.

GPT4:

The purpose of Machine Learning.
["machine learning", "computer", "make decisions", "program", "learn", "data", "patterns", "predict"]
["Machine_learning", "Computer", "Decision-making", "Computer_program", "Learning", "Data", "Pattern_recognition", "Prediction"]

G   Software

Our GNN was implemented using the Python library Pytorch-Geometric [Fey and Lenssen, 2019] and the reinforcement learning agent was trained with Tianshou [Weng et al., 2022]. The keyword extraction process was implemented using Langchain [Chase, 2022].