Knowledge Tracing Using Deep Learning Methods

Ghodai Mohamed Abdelrahman Mohamed Zaki

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science
The Australian National University

June 2023

Certificate Of Authorship/Originality

I, Ghodai M. Abdelrahman, declare that this thesis is submitted in fulfilment of the requirements for the degree of Doctor of Philosophy, in the School of Computing at the Australian National University.

I certify that this thesis is my own original work, except where otherwise indicated. I also certify that this document has not been submitted for obtaining an award at any other academic institution.

Ghodai Mohamed Abdelrahman Mohamed Zaki
23 June 2023

Dedicated to my dear parents and my dear husband.

Acknowledgments

First and foremost, I submit my faithful gratitude to the all-mighty ALLAH for enabling me to finish this work and blessing me with mercy, support, and guidance during this Ph.D. journey.

I would like to express my deepest appreciation to my principal supervisor, Professor Qing Wang, for her constant guidance, support, and encouragement throughout my research. I was very honored to have a distinguished supervisor with deep insight, critical thinking, and wisdom, who taught me valuable knowledge and research skills and instilled a professional work ethic.

Also, I want to sincerely thank my wonderful co-supervisor Dr. Bernardo Nunes for his continuous support and guidance during my journey.

I dedicate this thesis to the memory of Dr. Yu Lin who was a great supporter, educator, and a source of helpful academic advice.

Last but not the least, I would like to thank my family: my parents, my husband Dr. Sherif Gad, and my beloved daughter Joury for supporting me spiritually throughout writing this thesis and generally in my life.

I want to thank my amazing friends and colleagues whom I was so lucky to have at ANU.

Finally, I want to thank ANU for offering me a life-changing scholarship to pursue my Ph.D. research in a such world-class research environment.

Publications

The contributions from the work presented in this thesis have been published across multiple peer-reviewed journals and conferences. A list of publications in chronological order is given below:

(1) G. Abdelrahman and Q. Wang. "*Knowledge Tracing with Sequential Key-Value Memory Networks*", The 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2019.

Related work, research problem, methodology, experimental design, evaluation, and results analysis of the Sequential Key-Value Memory Networks (SKVMN) method are introduced in Chapter 4.

(2) G. Abdelrahman and Q. Wang. "*Deep Graph Memory Networks for Forgetting-Robust Knowledge Tracing*", IEEE Transactions on Knowledge and Data Engineering (TKDE), 2022.

Related work, research problem, methodology, experimental design, evaluation, and results analysis of the Deep Graph Memory Network ((DGMN) method are outlined in Chapter 5.

(3) G. Abdelrahman, Q. Wang, and B. Pereira Nunes. "*Knowledge Tracing: A Survey*", ACM Computing Surveys, 2023.

Background concepts, survey methodology, related work review, benchmarks overview, and discussion of the Knowledge Tracing Survey work are introduced in Chapter 3.

(4) G. Abdelrahman and Q. Wang. "*Learning Data Teaching Strategies Via Knowledge Tracing*", Knowledge-Based Systems, 2023.

Related work, research problem, methodology, experimental design, evaluation, and results analysis of Knowledge Augmented Data Teaching (KADT) method are presented in Chapter 6.

(5) G. Abdelrahman, S. Mohamed, Q. Wang and Y. Lin. "*DBE-KT22: A Knowledge Tracing Dataset Based on Database Systems Exercises*", Under Review.

Research contributions, data collection methodology, dataset description, and experimental evaluations of the DBE-KT22 dataset work are presented in Chapter 7

Abstract

This thesis aims to address a fundamental question relating to human and machine learning: *Can a machine effectively trace the dynamics of a student's knowledge states?* The thesis assumes a generic definition for a student, which can be either a human learner or a machine learning model. This fundamental question lies at the heart of the Knowledge Tracing (KT) problem, which has attracted increasing interest in recent years.

Addressing the KT problem is essential for a variety of applications in education, such as massive open online courses (MOOCs), intelligent tutoring systems, educational games, and learning management systems. Generally speaking, a KT model targets to achieve two main objectives. First, it needs to find effective representations of a student's knowledge states that can capture important learning concepts/skills in a learning course and their mutual relationships (e.g., prerequisite dependencies). Second, it aims to accurately trace temporal dynamics over a student's knowledge state representations which are conditioned on performance observations from their past exercise answering activities. From a machine learning perspective, a KT model is usually performed through a supervised sequence prediction task. In such tasks, a KT model sequentially predicts the probabilities of correctly answering recently observed questions conditioned on the past exercise answering history, usually considering the relationships among questions and learning concepts in the learning subject. Solving this sequence prediction task involves a number of challenges, including modeling long and short-term dependencies in an exercise answering history, finding effective ways to leverage dependency relationships among exercises, and dealing with students' forgetting behavior that may negatively impact their knowledge progress over time.

To address these challenges, this thesis makes the following research contributions. First, we present a comprehensive literature review of research conducted on existing KT models and the relevant datasets and benchmarks used for model evaluation. Then, we address the challenge of modeling long-short term dependencies in exercise answering sequences by proposing a novel deep sequence learning model that can dynamically adjust its temporal attention to focus on relevant exercises for answer prediction. Following that, we consider the effect of a student's forgetting behavior during knowledge state estimate through a novel forgetting-robust model that tracks forgetting features over learning concepts during the exercise sequence modeling. We highlight the potential of KT models in evolving effective teaching strategies through a novel reinforcement learning framework, which can leverage knowledge state representations to optimize a teacher agent interactively. We comprehensively evaluate our proposed models on well-established KT datasets. Furthermore, we propose a new KT dataset that addresses existing KT datasets' limitations and make it publically available. Finally, we outline future research directions and possible application areas in the KT domain.

Contents

List of Figures

List of Tables

Notation and Terminology

Notation

X	Exercise history
q_t	Question tag
y_t	Binary variable representing the answer
P_t	Predicted correct answer
C, KC	Learning concepts
t	Time step
Q	Set of question tags
k_t, v_t	Embedding vectors
d_k,d_v	State dimensions
w_t	An attention vector
A , **B**	Embedding matrices
$\mathbf{M^k}$	Key matrix
$\mathbf{M^v}$	Value matrix
N	Number of learning concepts
r_t	Read vector
f_t	Summary vector
d_t	Identity vector
e_t	Erase signal
a_t	Addition signal
W	Weight matrix
$\delta(.)$	One-hot encoding vector
G	Undirected and weighted graph
$\hat{\mathbf{A}}$	Adjacency matrix with self-loops
I	Identity matrix
$\hat{\mathbf{D}}$	Diagonal degree matrix
b	Bias
$\mathbf{F}_t$	Forgetting feature matrix
gw_t	Forget gating vector
$\hat{D}^{train}$	Training mini-batch
$\mathcal{L}$	Loss function
S	State space
A	Action space
T	State transition function
R	Reward function

Terminology

AFM	Additive Factor Model
AKT	Attentive Knowledge Tracing
AdaptKT	Adaptable Knowledge Tracing
BKT	Bayesian Knowledge Tracing
CKT	Convolutional Knowledge Tracing
CoKT	Collaborative Knowledge Tracing
DKT	Deep Knowledge Tracing
DKVMN	Dynamic Key-Value Memory Networks
DBN	Dynamic Bayesian Network
DGMN	Deep Graph Memory Network
E2EMem	End-to-End Memory Network
EERNN	Exercise-Enhanced Recurrent Neural Network
EKT	Exercise-Aware Knowledge Tracing
GRUs	Gated Recurrent Units
GKT	Graph-based Knowledge Tracing
GIKT	Graph-based Interaction Knowledge Tracing
HMM	Hidden Markov Model
ITS	Intelligent Tutoring System
IRT	Item Response Theory
KT	Knowledge Tracing
KV-MemNN	Key-Value Memory Network
KTM	Knowledge Tracing Machine
KPT	Knowledge Proficiency Tracing
KADT	Knowledge Augmented Data Teaching
LSTM	Long Short Term Memory
LPKT	Learning Process-consistent Knowledge Tracing
MOOCs	Massive Open Online Courses
MANNs	Memory-Augmented Neural Networks
MemNNs	Memory Networks
MDP	Markov Decision Process
NTM	Neural Turing Machine
PFA	Performance Factor Analysis
PEBG	Pre-training Embeddings via Bipartite Graph
QA	Question Answering
RNN	Recurrent Neural Network
RL	Reinforcement Learning
RKT	Relation-Aware Self-Attention for Knowledge Tracing
SKVMN	Sequential Key-Value Memory Network
SAKT	Self-Attentive Knowledge Tracing
SAINT	Separated Self-AttentIve Neural Knowledge Tracing
SKT	Structure-based Knowledge Tracing

Chapter 1

Introduction

This chapter introduces the work presented in this thesis by highlighting essential aspects of our research. We first provide an overview of the knowledge tracing (KT) domain and its significance in intelligent tutoring systems in Section 1.1. We outline the motivation points demonstrating the potential of KT methods in Section 1.2. We introduce the research objectives and challenges in Section 1.3. We briefly describe our research's contributions and workflow in Sections 1.4 and 1.5. Finally, we outline the remainder of the thesis's chapters in Section 1.6.

1.1 Overview of Knowledge Tracing

Teaching is a vital activity to facilitate the transfer of knowledge. It is well-known that one key factor of teaching is the ability of human teachers to track the learning progress of their students. This ability allows human teachers to adjust their teaching pace, materials, and methodology to maximize the knowledge growth of each individual student. Over the past 30 years, a variety of online education platforms such as Massive Open Online Courses (MOOCs) [176, 168], intelligent tutoring systems [138, 34], and educational games [107] have emerged to complement and sometimes completely replace conventional education systems. The COVID-19 pandemic, for example, has challenged conventional classroom-based teaching and sped up digital transformation in education systems. To alleviate the disruption of COVID-19, teachers, and students worldwide had to adjust to an online education mode rapidly. However, while there is a pressing need for teaching using computer technologies, technology-enhanced teaching has also posed new challenges. One of such challenges is to determine *how to effectively track the learning progress of a student through their online interaction with teaching materials* – known as the *Knowledge Tracing* (KT) problem [5, 179]. Generally speaking, knowledge tracing aims to observe, represent, and quantify a student's knowledge state, e.g., the mastery level of skills underlying the teaching materials.

To better understand the KT problem, let us consider the learning activity depicted in Figure 1.1. The figure shows an interaction scenario between a student and an Intelligent Tutoring System (ITS) in which the student is given a sequence of questions from a question set $\{q_1, q_2, q_3, q_4\}$ and asked to answer these questions. During the interaction, the ITS estimates the student's knowledge states over the skills $\{k_1, k_2, k_3, k_4\}$ (e.g., math skills such as addition, subtraction, and multiplication) that are required to answer these questions. It is worth noting that skills are often referred to using different terms in the KT literature, such as *learning con-*

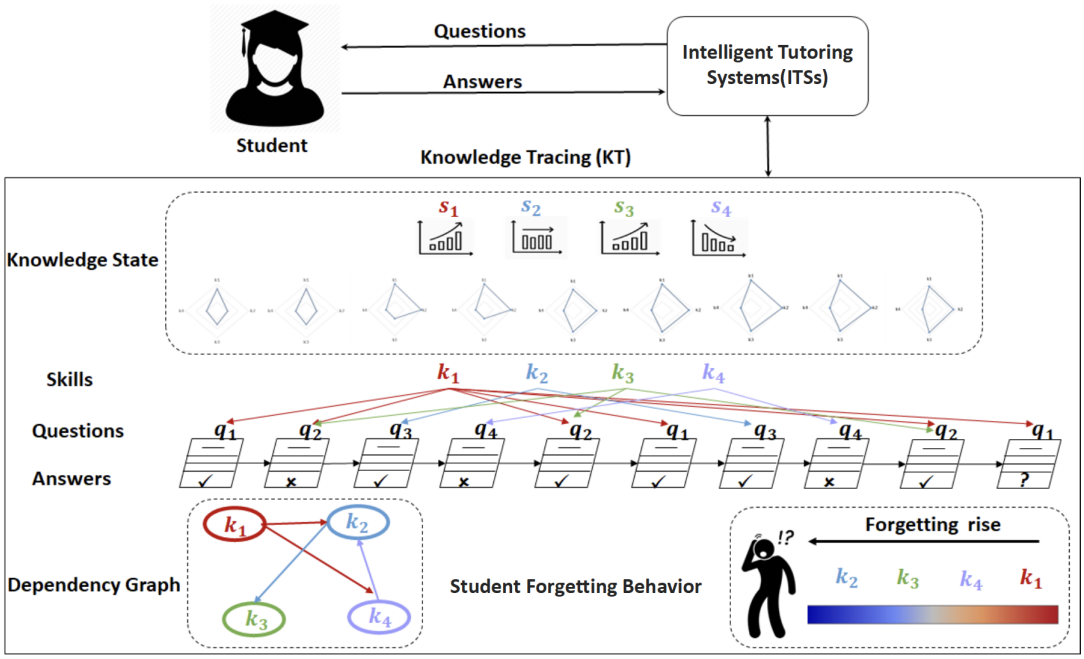


Figure 1.1: An example scenario for knowledge tracing in an Intelligent Tutoring System (ITS).

cepts, *knowledge components*, and *latent concepts* [78, 1, 201]. It is worth noting that *skill* is also referred to as *knowledge component* in some previous studies [102].

Tracing a student's knowledge state is a challenging task due to several reasons:

- Each question might require more than one *skill*, which adds complexity to trace knowledge states. For instance, as shown by the arrows going from skills to questions in Figure 1.1, question q_2 requires the skills $\{k_1, k_3\}$.

- Dependency among skills is another important factor to consider when tackling the KT problem. For example, although q_3 requires only skill k_2, skills k_1 and k_4 are prerequisites for skill k_2 according to the dependency graph shown in Figure 1.1. Thus, the mastery level of skills required by the question q_3 should also consider k_1 and k_4, in addition to skill k_2.

- A student's forgetting behavior [39] may result in decaying their knowledge over skills. By modeling forgetting features, skills can be ranked by their relevance to forgetting. For example, the bottom of Figure 1.1 shows that skill k_1 is least affected by forgetting when the latest question q_1 is reached, whereas skill k_2 is the most affected one.

1.1.1 A Historical View of Knowledge Tracing

Historically, the notion of knowledge tracing was introduced by Anderson et al. in a technical report [5] for cognitive modeling and intelligent tutoring in 1986, which was later published

in the *Artificial Intelligence* journal in 1990 [5]. Since then, many attempts have been made to design machine learning models for solving the KT problem. Early attempts [179, 33] followed Bayesian inference approaches, which usually relied on oversimplifying the model assumptions (e.g., assuming only one skill) to make the posterior computation tractable. Later, with the rise of classic machine learning methods such as logistic regression models [191], another direction followed by KT is to use parametric factor analysis approaches which trace a student's knowledge states and perform the answer prediction based on modeling a variety of factors [20, 21, 134], including (1) aspects about students such as prior knowledge, learning capacity, or learning rate; (2) aspects about learning materials such as familiarity, the number of previous practices, or difficulty; (3) aspects about a learning environment itself such as the nature of the learning channel (paper- or computer-based) and the temporal context of the practice time (within an examination period or a regular study period). In addition to these, psychological studies about the learning behavior [119] and forgetting behavior [74] of students have also suggested additional factors, such as the time lapse between a student's different interactions and the number of times practicing learning materials, to be considered when tracing knowledge states. It is worth noting that this direction of KT is still active [178, 44] and is considered as an alternative to the recent state-of-the-art approaches based on deep learning.

Motivated by breakthroughs achieved by deep learning techniques [92], deep learning KT models have emerged rapidly. Piech et al. [137] has pioneered this research direction and revealed the power of deep learning techniques for knowledge tracing. They proposed a model called Deep Knowledge Tracing (DKT), which applies Recurrent Neural Networks (RNNs) [98] to capture temporal dynamics in a sequence of interactions between questions and answers by a student, and based on that, to predict the student's answer for a new question. Empirical results showed that DKT outperformed traditional KT models on several benchmark datasets. This attempt highlighted the potential of using deep learning models in addressing the KT problem. In recent years, an increasing number of studies have exploited the development of deep learning KT models from different perspectives, including:

- **Memory structures.** Inspired by memory-augmented neural networks [52, 114], deep learning KT models are extended by augmenting more powerful memory structures, typically key-value memory, for capturing knowledge states dynamically at a finer granularity such as the mastery level of each individual skill (e.g. [201, 1]).

- **Attention mechanisms.** Inspired by the *Transformer* architecture [177] and some further developments in natural language processing applications, attention mechanisms are incorporated into deep learning KT models for capturing the relationships among questions and their relevance to a student's knowledge states (e.g., [126, 45, 26, 157, 127]).

- **Graph representation learning.** Inspired by the representational power of graph learning techniques such as Graph Neural Networks [149, 80], deep learning KT models are equipped with graph learning techniques to leverage the rich structural information from graphs that can flexibly model relationships among questions and skills (e.g. [122, 195, 173]).

- **Textual features.** Question text may potentially contain a wealth of information, such as skills required by questions, the difficulty of questions, and relationships between questions. Several deep learning KT models leverage textual features from question text for learning question representations and tracing a student's knowledge states (e.g. [198, 161, 101]).

- **Forgetting features.** Motivated by the *learning curve theory* [119], a recent trend in developing deep learning KT models is to incorporate forgetting features so that a student's forgetting behavior can be taken into consideration for knowledge tracing (e.g. [121, 24, 2]).

These studies facilitate the translation of breakthroughs in deep learning techniques into the KT domain.

1.1.2 A Mathematical Formulation for Knowledge Tracing

From a machine-learning perspective, the knowledge tracing (KT) problem can be generally represented as predicting the likelihood of correctly answering an exercise conditioned on a student's past exercise answering history (i.e., question-answer pairs) [31]. A mathematical formulation of the KT problem is outlined as follows.

Let $Q = \{q_1, \ldots, q_{|Q|}\}$ be the set of all distinct question tags in a dataset. Each $q_i \in Q$ may have a different difficulty level, which is often not explicitly provided. An *exercise* x_i is a pair (q_i, y_i) consisting of a question tag q_i and a binary variable $y_i \in \{0, 1\}$ representing the answer, where 0 means that q_i is incorrectly answered and 1 means that q_i is correctly answered. When a student interacts with the questions in Q, for t time steps, a history of exercises $X = \langle x_1, x_2, \ldots, x_{t-1} \rangle$ undertaken by the student can be observed. Moreover, questions in Q are associated with a set $C = \{c_1, c_2, \ldots, c_N\}$ of learning concepts (e.g., skills in a course).

Conditioned on an answering history X, we want to estimate the probability $p\,(y_t = 1 | q_t, X)$ of correctly answering a question at the time step t.

1.2 Research Motivation

Besides the large potential economic impacts of enhancing the fast-growing online education industry [111], we are motivated by the social and developmental impacts that can be achieved by equipping this industry with automated knowledge tracing models. The student base for such online systems is enormous, with students from almost everywhere. This means that any effective enhancement would impact millions of people worldwide. In addition, online ITSs provide a more cost-efficient alternative with high-quality learning materials for students who cannot afford conventional education expenses. Adaptive ITSs with an ability to effectively trace a student's knowledge would lead to a better-customized learning experience and, therefore, better student learning outcomes.

Additionally, conventional education institutions such as universities can benefit from automated knowledge tracing models by reducing teaching efforts in customizing the learning experience for different student groups. Consequently, the expenses of the education process

can be reduced while the overall educational experience is enhanced. In summary, effectively addressing the KT problem can lead to the following:

- More effective and personalised learning process that can fit students' different needs;
- Better learning materials recommendations informed by a student's knowledge state;
- Assisting instructors with automated algorithms for course design and evaluations;
- Increase of trust in the effectiveness of automated learning systems.

1.3 Research Challenges and Objectives

Knowledge tracing demands modeling sequential exercise answering data while effectively leveraging additional sources of useful information, such as relationships between questions and learning concepts in the task. This raises many research challenges in dealing with long-short term dependencies in the exercise answering sequences besides counting for psychological human aspects that impact a student's knowledge growth. In this thesis, we consider four main research challenges as follows:

- **Challenge 1**: How to effectively represent and track a student's knowledge state given past exercise sequences?
- **Challenge 2**: How to represent and model a student's forgetting behavior for robust knowledge tracing?
- **Challenge 3**: Could teaching strategies be learned via knowledge tracing?
- **Challenge 4**: What are the current limitations in KT datasets? How to address them?

To address these mentioned challenges, in this thesis, we aim to achieve the following research objectives:

- **Objective 1**: We target to develop knowledge representation models that can effectively capture a student's knowledge state given the past exercise answering sequences and relationships between questions and learning concepts in the task.
- **Objective 2**: We aim to design robust knowledge state representation to a well-known cognitive effect of knowledge forgetting due to human memory temporal decay [74] that negatively impacts the probability of correctly answering a given question.
- **Objective 3**: We propose a novel interactive reinforcement learning method that utilizes a knowledge tracing model for state representation in a learning material teaching strategy for a student model.
- **Objective 4**: We create and publish a new KT dataset that mitigates the limitations in the existent KT datasets and facilitates new research directions.

1.4 Contributions

To target the defined research objectives, we propose multiple points of novel contributions, including a) effectively modeling long-short term dependencies across a student's exercise answering sequence, b) accounting for a student's forgetting behavior as one of the important psychological effects that impact their knowledge growth during the knowledge state modeling, c) proposing ways to leverage relational dependencies across questions and learning concepts in the form of graphs to enhance the accuracy of knowledge state estimation, d) designing an effective method to evolve teaching strategies guided by knowledge tracing in an interactive manner, and e) address the existent limitations in KT datasets by proposing a new dataset collected from real student exercise answering. We detail each of these research contributions points as follows:

1. We provide a comprehensive survey of the knowledge tracing literature. We cover a broad range of methods, from early attempts to the recent state-of-the-art methods, while highlighting the theoretical aspects of models and the characteristics of benchmark datasets. Besides, we shed light on key modeling differences between closely related methods and summarize their application assumptions.

2. We propose a novel KT model called *Sequence Key-Value Memory Network (SKVMN)* that can effectively represent the knowledge state as a dynamic multi-dimensional tensor over learning concepts and model the long short-term dynamics of the exercise sequence. This is achieved through the utilization of memory-augmented deep neural networks [53, 148, 147]. In addition, we propose a sequence prediction module in our model that utilizes a novel long short-term memory (LSTM) cell called *hop-LSTM*. Our proposed *hop-LSTM* enhances the conventional LSTM cell by identifying relevant sub-sequences of questions that share temporal dependencies and modeling each subsequence separately to prevent irrelevant updates to the cell's hidden state. This contribution addresses research challenge 1.

3. We propose a novel knowledge tracing model named *Deep Graph Memory Network* (DGMN) that incorporates a forget-gating mechanism to capture students' forgetting behaviors dynamically during the knowledge tracing process. Particularly, this forget-gating mechanism is built upon attention to forgetting features over learning concepts considering their mutual dependencies. Further, this model can learn relationships between learning concepts from a dynamic concept graph in light of a student's evolving knowledge state. This contribution addresses research challenge 2.

4. We propose a novel method called *Knowledge Augmented Data Teaching* (KADT), which can optimize a data teaching strategy for a machine learning student model by tracing its knowledge progress over multiple learning concepts in a given supervised learning task. Specifically, the KADT method incorporates a knowledge tracing model to dynamically capture the knowledge progress of a student model in terms of essential learning concepts. Then, we utilize the knowledge state over defined class labels as a state representation for a reinforcement learning agent acting as a teacher model. This contribution addresses research challenge 3.

5. We perform a comprehensive review of the existent knowledge tracing datasets and highlight their limitations. Further, we propose a new knowledge tracing dataset named *Database Exercises for Knowledge Tracing* (DBE-KT22) that addresses the identified limitations. DBE-KT22 is collected from real-world learning exercise activities from an ANU undergraduate course on database management systems. This contribution targets answering research challenge 4.

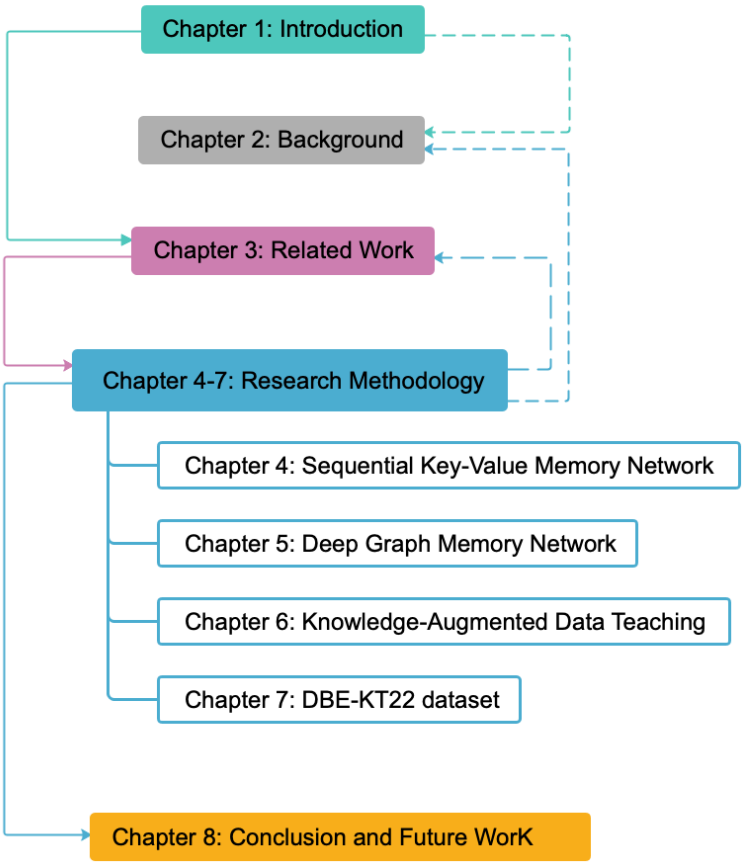


Figure 1.2: Research workflow followed in this thesis.

1.5 Research Workflow

Our workflow for achieving the research objectives is depicted in Figure 1.2, as we further discuss in this section.

1. Introduction: Presenting our research domain and introducing the research motivation, objectives, challenges, and contributions.

2. Background: We obtained the necessary background knowledge related to KT and general machine learning, recurrent neural networks and their variants for sequence modeling, memory-augment neural networks, Graph Neural Networks, Convolutional Neural Networks, and Reinforcement Learning.

3. Related Work: We studied the existing literature on knowledge-tracing methods over two broad categories, including probabilistic knowledge-tracing methods and deep knowledge-tracing methods. It was an ongoing study throughout our research, which helped refine our research problems, design prototypes, and evaluate their evaluation.

4. Research Methodology: Proposing three novel methods to address our research questions. Moreover, we propose a new dataset named DBE-KT22 that addresses limitations in existing KT datasets. Our dataset is collected from a real relational databases course taught at the Australian National University (ANU) in Australia for undergraduate and graduate students.

5. Conclusion and Future Work: Concludes and summarizes the research contributions and discusses future research directions and possible application areas.

1.6 Thesis Outline

The remainder of this thesis is organized as follows:

- Chapter 2 introduces the underlying concepts related to the knowledge tracing domain and presents a primer on relevant deep learning models.
- Chapter 3 overviews the related research work and existing datasets in the knowledge tracing domain over two broad categories, including probabilistic knowledge tracing methods and deep knowledge tracing methods. Moreover, it highlights the research gaps in existing methods.
- Chapter 4 introduces and evaluates our Sequential Key-Value Memory Networks (SKVMN) method that addresses research challenge 1.
- Chapter 5 introduces and evaluates our Deep Graph Memory Networks (DGMN) method that addresses research challenge 2.
- Chapter 6 introduces and evaluates our Knowledge-Augmented Data Teaching Strategies (KADT) model that addresses research challenge 3.
- Chapter 7 presents our proposed DBE-KT22 dataset that addresses research challenge 4.
- Chapter 8 concludes and summarizes the research contributions and indicates future research work.

Chapter 2

Background

In this chapter, we present the background related to the methods proposed in this thesis. We first introduce intelligent tutoring system (ITS) which is one common application domain for knowledge tracing in Section 2.1. In Section 2.2, we introduce the learning curve theory that lays the foundations to understand how human memory retention and decay impact knowledge acquisition during learning. After that, we provide a review of deep learning methods for sequence modeling, including recurrent neural networks in Section 2.3, memory-augmented neural networks in Section 2.4, and graph neural networks in Section 2.5. Finally, we conclude this chapter in Section 2.6.

2.1 Intelligent Tutoring System

An intelligent tutoring system (ITS) aims at providing customized learning guidance to students without intervention from human teachers [138, 142]. Figure 2.1 provides a modular design for the four main components in an ITS.

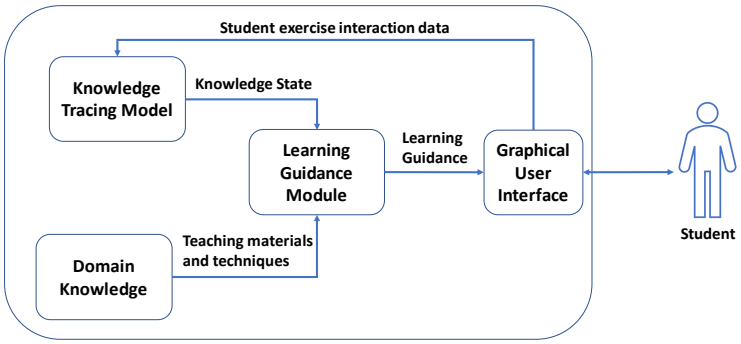


Figure 2.1: A modular design diagram for an intelligent torturing system.

First, the graphical user interface (GUI) handles interaction with a student such as recording answers to exercises or presenting learning materials. Second, the knowledge tracing model is responsible for tracking the current knowledge state of a student on different learning concepts (e.g., topics in a math course) given the past interaction history. Third, the domain knowledge component executes different teaching strategies and techniques to teach learning

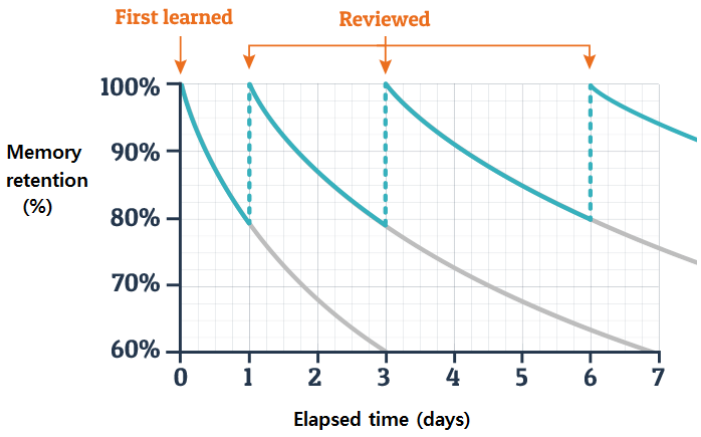


Figure 2.2: Ebbinghaus forgetting curve over time with and without the impact of reviewing the learned information [28].

materials to a student. Finally, the learning guidance module utilizes information from the current knowledge state and domain knowledge to customize the teaching strategy for a student.

It is a vital success factor for any ITS to have an effective knowledge tracing capability in order to provide a similar learning experience that a human teacher can provide for students [31, 137, 36, 24, 178].

2.2 Learning Curve Theory

Attempts to understand the memory working mechanisms in the human brain go back almost a century ago with the work done by Hermann Ebbinghaus [39, 120], who studied learning and forgetting forces that affect information retention from an experimental psychology perspective to formulate what is currently known as the learning curve theory [119]. The forgetting force results in the decay of memory retention over time. Ebbinghaus indicated that this decay over time follows an exponential decay pattern as shown in Equation 2.1:

$$R = e^{-\frac{t}{S}} \tag{2.1}$$

where R is the memory retention rate, t is the time steps passed since being first introduced to the information, and S is the memory stability in holding learned information in lack of revision and practice over time.

The learning force opposes the impact of the forgetting force through repeated revision and practice of the learned information. Figure 2.2 shows this interplay between the forgetting and learning effects over time. While Ebbinghaus's work is seminal in the literature of studying student forgetting behavior, a recent study [140] suggests that human memory forgetting and retention do not follow a single time-decaying function but involve multiple phases that impact the memory consolidation in parallel.

Tracing the knowledge state of a student without counting on the impact of the forgetting effect can lead to a significant error in the estimate and thus mislead the actions taken by an

intelligent teaching strategy. Modeling the forgetting effect during knowledge tracing is one of the major challenges that the literature aimed to tackle. We introduce these attempts in detail in Section 3.

2.3 Sequence Modelling

Over the last decade, deep neural network models achieved major breakthroughs in sequence modeling applications [151, 163, 51, 48, 50]. This made many knowledge tracing methods adopt them for representing and modeling the temporal dynamics in a student's knowledge states conditioned on exercise answering sequences. In this section, we provide an overview of different types of deep neural networks for sequence modeling.

2.3.1 Recurrent Neural Networks

Conventional neural networks assume that input samples are independent and identically distributed (iid) [89], which is a valid assumption with non-temporal data scenarios such as classifying images of cats and dogs. However, sequence data does not satisfy this assumption as the latest observation depends on an exercise answering sequence. For example, consider a sentiment analysis task [128], in which we need to understand the human's opinion from a paragraph of text. We cannot deal with each word separately to achieve such an objective. Similarly, in the knowledge tracing problem, we cannot assume that a student's answers are independent of each other as the probability of answering the latest exercise may be dependent on the performance of answering previous relevant exercises.

Alternatively, RNNs do not assume that input samples are iid and aim at modeling sequence-level dynamics by allowing information from previous time steps to be preserved during the processing of the latest input samples. At the core of RNNs, this is achieved through internal recurrence loops on each hidden unit. Therefore, the output of a hidden unit can flow to its input channel to represent past sequence dynamics. This recurrence gives RNNs a form of memory that can hold important information on the sequence level.

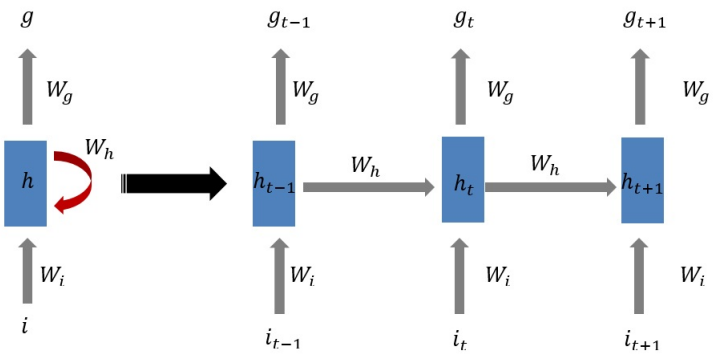


Figure 2.3: A recurrent neural network and its unfolding over time.

Figure 2.3 illustrates a recurrent neural network. The variable i represents an input, which

can be a one-hot encoding vector representing the current word position in the sequence. The variable h represents the network's hidden state, and the variable g represents the network's output, which can be another one-hot encoding vector indicating the next word in the sequence. The learning parameters W_i, W_h, and W_g represent the weight vectors for the input, hidden state, and output respectively. The hidden state and output are calculated as follows:

$$h_t = \sigma(W_i \cdot i_t + W_h \cdot h_{t-1}) \tag{2.2}$$

$$g_t = \text{Softmax}(W_h \cdot h_t) \tag{2.3}$$

where $\sigma(i) = \frac{1}{1+e^{-i}}$ and $\text{Softmax}(i_k) = \frac{e^i k}{\sum_{j=0}^{k} e^i j}$ are activation functions.

2.3.2 Long Short-term Memory

Despite the effectiveness of RNNs in modeling sequence dynamics, their ability to model long-term dynamics over long sequences is quite limited [95, 133]. To overcome this limitation, long short-term memory networks were introduced [64]. LSTM has two hidden states instead of one as in RNN.

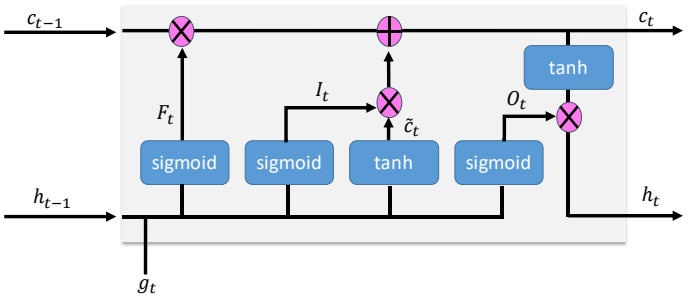


Figure 2.4: A visual representation for the design of an LSTM cell.

A LSTM cell has three gates I_t, F_t, and O_t in addition to two states h_t and v_t. The input gate I_t determines how much information from the current input g_t is going to affect the current cell's state c_t. The forget gate F_t decides what information should be forgotten or kept from the current input g_t and the previous cell's state c_{t-1}. The output gate o_t controls which information from the current cell's state c_t should flow to the current hidden state h_t. The cell's state c_t stores long and short-term information about the sequence. A hidden state h_t is the cell's output with respect to the current input g_t, the current cell's state c_t, and the previous hidden state h_{t-1}. We present the corresponding equations for a LSTM cell as follows. Figure 2.4 depicts the design of a LSTM cell.

$$I_t = \sigma(\mathbf{W}_\text{i}[\text{h}_{\text{t}-1}, \text{g}_\text{t}] + \text{b}_\text{i}) \tag{2.4}$$

$$F_t = \sigma(\mathbf{W}_\text{f}[\text{h}_{\text{t}-1}, \text{g}_\text{t}] + \text{b}_\text{f}) \tag{2.5}$$

$$O_t = \sigma(\mathbf{W}_\text{o}[\text{h}_{\text{t}-1}, \text{g}_\text{t}] + \text{b}_\text{o}) \tag{2.6}$$

$$\tilde{c}_t = \text{Tanh}(\mathbf{W}_c[h_{t-1}, g_t] + b_c) \tag{2.7}$$

$$c_t = F_t \odot c_{t-1} + I_t \odot \tilde{c}_t \tag{2.8}$$

$$h_t = O_t \odot \text{Tanh}(c_t) \tag{2.9}$$

where $\sigma(i) = \frac{1}{1+e^{-i}}$ and $\text{Tanh}(u_i) = \frac{e^{u_i}-e^{-u_i}}{e^{u_i}+e^{-u_i}}$ are activation functions.

2.3.3 Gated Recurrent Unit

A gated recurrent unit is a simplified version of LSTM, aiming at speeding up the gating computations while preserving the same performance level [29]. Figure 2.5 shows a modular design for a gated recurrent unit. Instead of three gates as in LSTM, GRUs have only two gates. First, the update gate z_t updates the hidden state conditioned on the input and the past hidden state values. Second, the reset gate r_t controls how much information to forget from the hidden state conditioned on the input and the past hidden state values. The following equations provide the calculations for these gates.

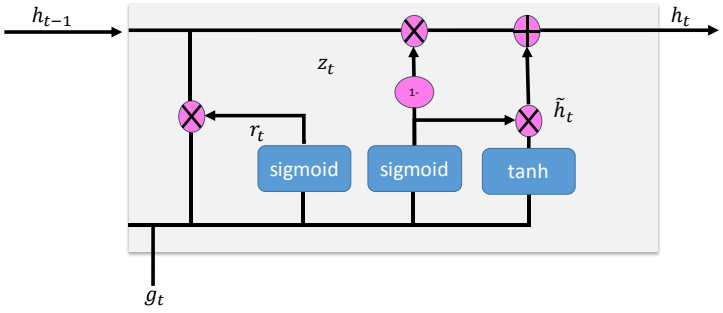


Figure 2.5: A modular design for gated recurrent units.

$$z_t = \sigma(\mathbf{W}_z[h_{t-1}, g_t]) \tag{2.10}$$

$$r_t = \sigma(\mathbf{W}_r[h_{t-1}, g_t]) \tag{2.11}$$

After calculating how much to update and forget from the hidden state, the new state value can be calculated in two steps:

$$\tilde{h}_t = \text{Tanh}(\mathbf{W}_h[r_t \odot h_{t-1}, g_t]) \tag{2.12}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.13}$$

where $\sigma(i) = \frac{1}{1+e^{-i}}$ and $\text{Tanh}(u_i) = \frac{e^{u_i}-e^{-u_i}}{e^{u_i}+e^{-u_i}}$ are activation functions. The first step removes from the hidden state what we need to forget, while the second step adds the needed updates to it.

2.3.4 Embedding and Attention Mechanisms

Neural networks are effective in learning with numeric representations such as real numbers [117]. However, working with alphanumeric representations such as words can be challenging. This is mainly due to optimization techniques that are based on the numerical representation of an input for gradient computations and error propagation. Moreover, numerical representations can be clustered to represent dependencies and similarities, but this is not a straightforward task when dealing with alphanumeric representations such as words. In the KT problem, we deal with alphanumeric inputs representing questions and learning concepts text tags. Therefore, exploring relevant methods for representing such input is vital from a KT perspective.

Word embedding techniques aim at transforming alphanumeric representations into numeric representations that satisfy two main characteristics. First, it has a lower dimensionality in comparison to the original representation. Second, it has to capture contextual similarities between words (i.e., vectors for words that usually appear with each other should be closer in the new representation space). Figure 2.6 shows three examples of graphical representations of word vectors. The lines represent word mathematical vectors. It can be noticed in Figure 2.6. (a) that it is possible to move from the word "Man" to the word "Queen" by replacing the word "King" with the word "Woman".

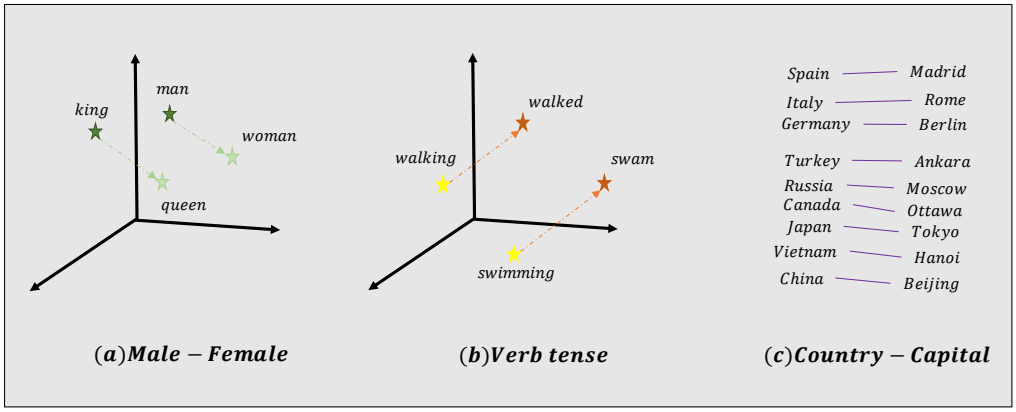


Figure 2.6: Examples for embedding vector representations for words. (a) A male-female mapping example. (b) A verb tense mapping example. (c) A country-capital mapping example.

In comparison to simple representation for text such as one-hot encoding (that is a binary representation with one on the relevant word place and zeros elsewhere) or bag of words (that is giving each word a unique numerical identifier and representing a text as a sparse matrix over a word vocabulary), word embedding vectors [113, 136] are learned through an optimization objective such as maximizing the log probability for predicting surrounding words in the text.

After obtaining an embedding representation for an alphanumeric input, we need to focus the learning process on relevant parameters that have the highest impact on the output given the current representation. Attention mechanisms [177] aim at fulfilling this requirement through learning a probability distribution over input embeddings reflecting their relative importance to the output. For example, Figure 2.7 shows a graphical representation of a recurrent neural

network used in a language translation task from English to French. It can be noticed in the upper left side of Figure 2.7 that the attention mechanism gives the highest weight for the "I" token when we generate the first equivalent token in the French language "Je".

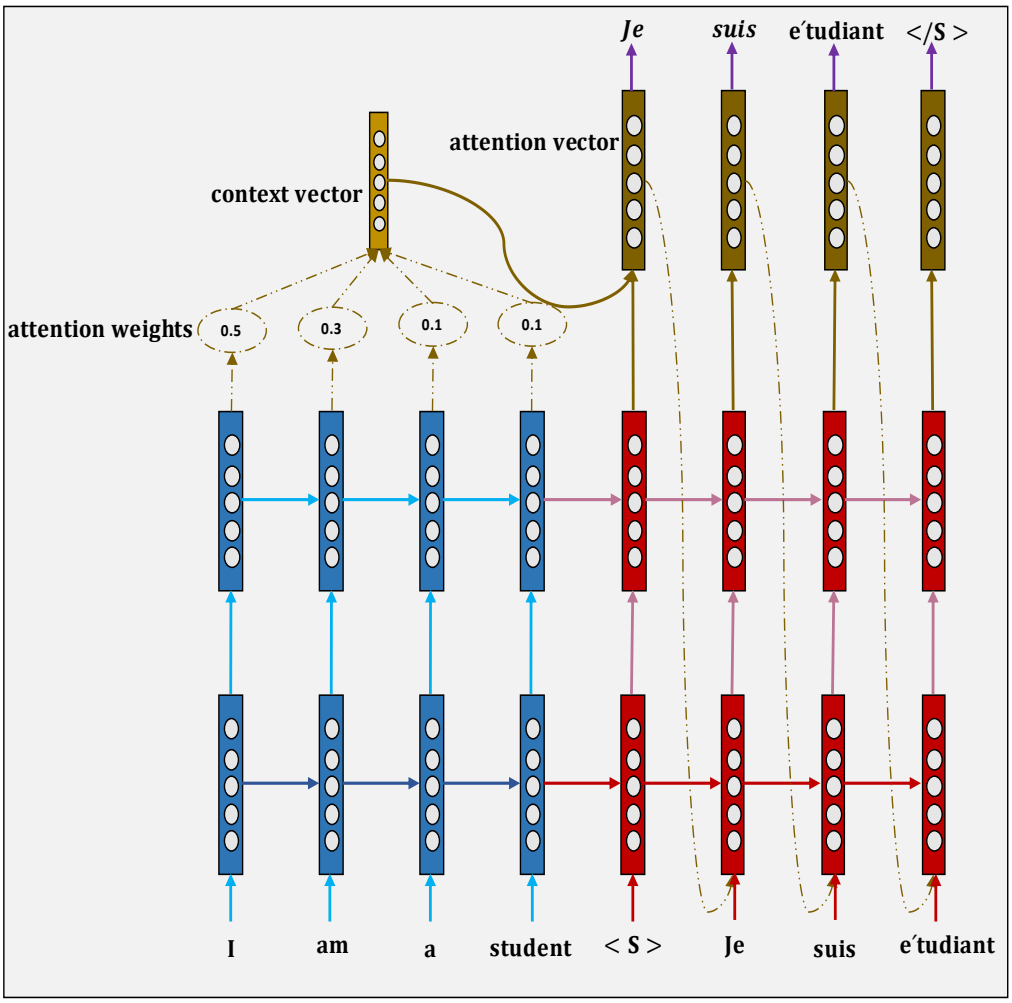


Figure 2.7: An example of an attention mechanism from a language translation task.

2.4 Memory-Augmented Neural Networks

Memory-augmented Neural Networks (MANNs) [148, 53] are a category of networks with external memory. External memory is generally a matrix containing input sequence information. A MANN often uses an attention mechanism to read the most relevant parts from an external memory for inference. MANNs are more capable models in comparison to RNNs due to the existence of larger memory structures. This helps when the input is too large to be properly embedded in the RNN fixed-size hidden memory vector. MANNs succeeded in many applications, including question answering (QA) [162, 86, 53], healthcare [58, 90], dialogue

system [188], and bioinformatics [88].

2.4.1 End-to-End Memory Networks

One of the first attempts in MANN models was the Memory Networks (MemNNs), proposed by [189]. Although it has good potential to address QA tasks, the model cannot be easily trained using backpropagation, and it requires a significant amount of supervision to prepare the training data carefully. For example, training signals were answered labels for not only each question but also facts supporting each answer. Thus, it does not apply to data in the form of pairs of inputs and labels. The End-to-End Memory Network (E2EMem) [162] is an extension of the MemNNs [189] model with the ability of end-to-end training on input label pairs. This ability results in flexibility in dealing with different learning tasks. The E2EMem process inputs across multiple layers before returning an answer. We first describe a single-layer and then introduce its multiple-layer version.

Figure 2.8 shows a single-layer E2EMem network that processes a sequence of inputs $X_1, \cdots, X_n$ and a query q to produce an answer a. For each sentence X_i, the query q and the answer a are a sequence of words coming from a vocabulary of size V. The model writes all inputs to an addressing memory $[m_1, \ldots, m_n]$ and a content memory $[c_1, \ldots, c_n]$.

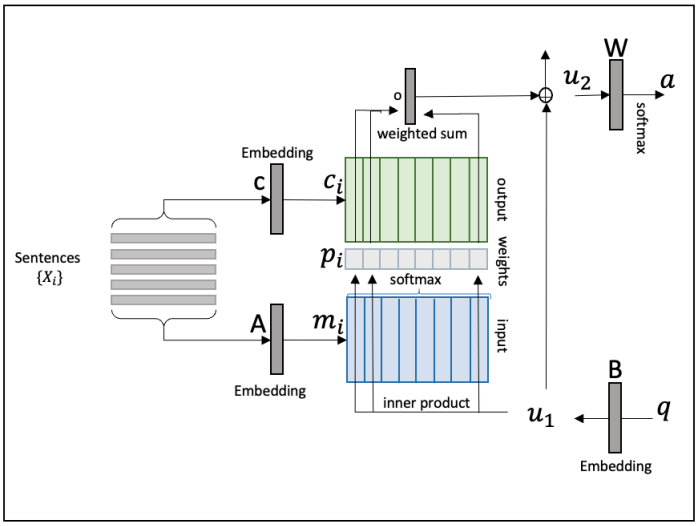


Figure 2.8: A single layer End-to-End Memory Network (E2EMem).

Addressing memory: Each sentence X_i is embedded into a continuous vector using an embedding matrix $\mathbf{A} \in \mathbb{R}^{d \times V}$ and is written to a memory slot $m_i \in \mathbb{R}^d$. The query q is also embedded into a vector u using another embedding matrix $\mathbf{B} \in \mathbb{R}^{d \times V}$.

Then, an attention vector p is obtained by applying the Softmax function to the inner product between the embedding vector u and the memory slot m_i:

$$\mathbf{p} = \text{Softmax}(\mathbf{u}^T \mathbf{m}_i) \tag{2.14}$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$. Conceptually, $\mathbf{p}$ measures the relevancy of input sentences

X_i to the query q. The dot product of u and m_i calculates the similarity between them in the embedding space.

Content memory: this memory is generated by embedding input sentences $X_1, \cdots, X_n$ using an embedding matrix $\mathbf{C} \in \mathbb{R}^{d \times V}$, where a memory slot c_i is an embeddeding vector of input sentence X_i. Relevant information from the content memory to the query embedding u is extracted using a summation vector o, which is obtained by an element-wise product between the attention vector p and the content memory slots.

$$o = \sum_{i=1}^{N} p_i c_i \tag{2.15}$$

Using the attention mechanism, the model can choose the most convenient contents from input to answer a question. After calculating the summation vector o, its element-wise summation and the input embedding u are then passed through a final weight matrix $\mathbf{W} \in \mathbb{R}^V$ and a softmax function is used to calculate the probability of each word being the correct answer:

$$\hat{a} = \text{Softmax}(W(o + u)) \tag{2.16}$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$. For optimizing the model's parameters, such as embedding matrices or weight matrices, a cross-entropy loss function between the predicted correct answer probability and the true answer is utilized.

To extend a single-layer model to a multi-layered variant, layers can be stacked on top of each other. In the E2EMem network of T layers, in each t^{th} layer, the model calculates the output u_{t+1} of the vector u_t, and the two memories m_i and c_i similarly to the single layer version. At the first layer, u_1 is the embedding of query q, and at the last layer, the output u_{T+1} is used to predict the answer. The embedding matrices can be shared across layers to reduce the number of parameters. In this case, the E2EMem is an RNN equipped with an external memory with a hidden state at each step t^{th} layer corresponding to the output u_{t+1}.

2.4.2 Neural Turing Machine

Another implementation of MANNs is the Neural Turing Machine (NTM) [52]. It can be considered as a different version of the Turing machine, resulting in an end-to-end training scheme with gradient descent. Figure 2.9 shows the NTM design architecture. NTM is fundamentally composed of a neural network called the controller, an RNN that reads inputs from external sources and interacts with memory through multiple readings and writing headers over multiple computational steps.

The memory is a matrix $\mathbf{M} \in \mathbb{R}^{N \times d}$, where N is the number of memory slots and d is the dimension of each slot. In the E2EMem network, the number of memory slots varies depending on the size of an input. Since each memory slot contains an input representation, the number of N slots in the NTM is fixed and independent of the input size.

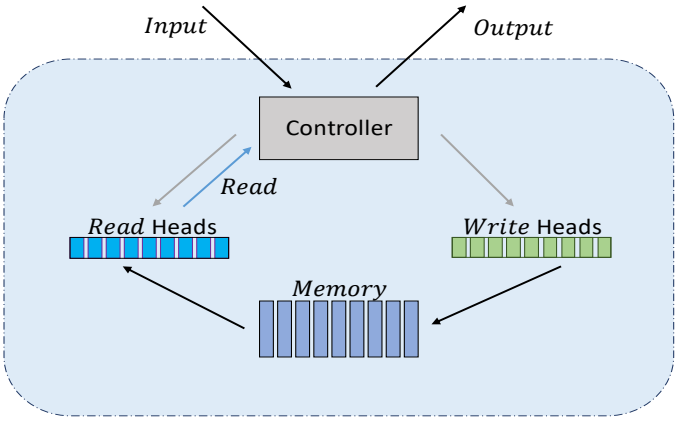


Figure 2.9: A Neural Turning Machine (NTM).

Read Process: Let $\mathbf{M}_t$ be the memory at time step t. At each time step, the controller retrieves information from memory through multiple reading heads, and each head reads a summation vector $r_t \in \mathbb{R}^d$ from memory using a weight vector $w_t^r = [w_1^r, \ldots, w_N^r]$:

$$r_t = \sum_{i=1}^{N} w_i^r \mathbf{M}_t(i) \tag{2.17}$$

where w_t^r is the reading weight measuring the contribution of a memory slot i to the summation vector. Here, w_t^r obeys the following constraints:

$$\sum_{i=1}^{N} w_i^r(i) = 1, \qquad 0 \leq w_i^r(i) \leq 1 \tag{2.18}$$

Write Process: After processing a summation vector r_t from the read heads, the controller writes back to the memory through multiple write heads with two steps: 1) erase step and 2) add a step. These steps are inspired by the input and forget gates in LSTM. For each write head, emanate a weight vector $w_t^r = [w_1^r, \ldots, w_N^r]$ along with an *erase vector* $e_t \in \mathbb{R}^d$, whose $\mathbf{M}$ elements lie in the range $(0, 1)$. The memory vectors $\mathbf{M}_{t-1}(i)$ from the previous time-step are modified as follows:

$$\tilde{\mathbf{M}}_\mathbf{t}(\mathbf{i}) = \mathbf{M}_{\mathbf{t-1}}(\mathbf{i})[1 - w_t(i)e_t] \tag{2.19}$$

The i^{th} memory slot is erased (i.e., set to zero) if both the corresponding locations in the attention vector w_t and the erase signal e_t are equal to 1; otherwise, the slot if left unchanged (i.e., multiplied by 1).

Then, the write head computes an add vector a_t, which is a representation vector of the input, and then it is added to every memory slot using the weight vector w_t^w:

$$\mathbf{M}_\mathbf{t}(i) = \tilde{\mathbf{M}}_\mathbf{t}(i) w_t(i) a_t \tag{2.20}$$

Since both vectors w_t^r for reading and w_t^w for writing are calculated in the same way, we

refer to both of them as w_t for notation simplicity. In an NTM, the weight vector w_t is computed by content-based addressing that takes into account the location information of the memory slots. The content-based system produces a normalized weighting w_t^c based on the similarity between the key vector k_t and the memory vectors $\mathbf{M_t}(i)$ with a positive key strength β_t.

$$w_t^c(i) = \frac{\exp(\beta_t K[k_t, \mathrm{M_t(i)}])}{\sum_j \exp(\beta_t K[k_t, \mathrm{M_t(j)}])} \tag{2.21}$$

The weight vector is then transferred as follows. First, it is multiplied by a weight from the previous step w_{t-1} based on the g_t gate. Second, it is rotated by a convolutional shift designed to focus on location information.

NTMs have some limitations. First, reading heads cannot extract information about the order of write operations, which may be important in sequential data. Second, there is no mechanism to prevent memory slots from overlapping with each other. Finally, memory slots that have been written cannot be freed, causing difficulties in storing long sequences.

2.4.3 Key-Value Memory Networks

The key-value memory network (KV-MemNN) [114] is an extension of end-to-end memory network (E2EMem), whose inputs are key-value pairs. Unlike the E2EMem network, which generates addressing and content memories of the same input sequence, KV-MemNN creates an addressing memory from the keys and a content memory from the values. Figure 2.10 shows the key-value memory network model for question answering.

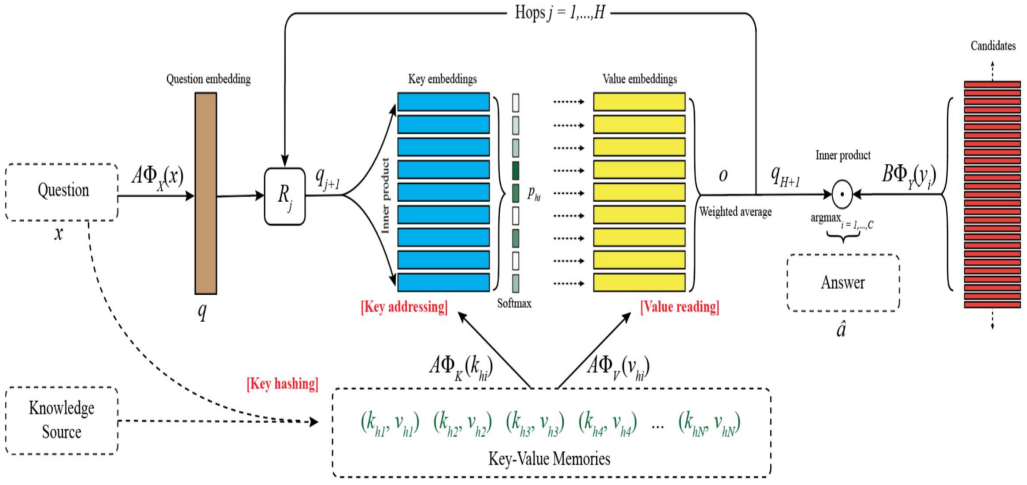


Figure 2.10: The Key-Value Memory Network (KVMN) model [114]

Given a question x, the KV-MemNN model's workflow goes over three steps: 1) key hashing, 2) key addressing, and 3) value reading.

Key hashing: The model uses a question to pre-select a small subset of key-value pairs $(k_t, v_t) \ldots (k_n, v_n)$ from a database that is relevant to the question. The selected keys share

at least one infrequent word with the question. Using infrequent words for comparison helps the model ignore common words (e.g., stop words) which do not contribute to the answer. It has to be noted that key hashing is not implemented in E2EMem.

Key addressing: Similar to the addressing memory in E2EMem, key addressing computes a weight vector using the addressing memory. During this step, the relevance probability of each candidate's memory is set by comparing the question with each key:

$$Ph_i = \text{softmax}(A\Phi_X(x) \cdot A\Phi_K(Kh_i)) \tag{2.22}$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$ and Φ is a feature map of dimension $\in \mathbb{R}^D$ and A is a $\in \mathbb{R}^{d \times D}$ matrix.

Value reading: Memory values are read by taking their weighted sum using addressing probabilities, and a vector o is returned.

$$o = \sum_i Ph_i A\Phi_V(Vh_i) \tag{2.23}$$

The memory access process is done by the "controller" neural network using $q = A\Phi_X(x)$ as a query. After obtaining the result o, the query is updated using $q_2 = R_1(q + o)$, where $\mathbf{R} \in \mathbb{R}^{d \times d}$. Addressing and reading steps are repeated using new R_i matrices to retrieve more pertinent information in subsequent access. After a fixed number of hops H, the state of the controller is used to compute a final prediction:

$$\hat{a} = \text{argmax}_{i=1,\ldots,C}\text{Softmax}(q_{H+1}^{\mathrm{T}} B\Phi_Y(y_i)) \tag{2.24}$$

The KV-MemNN model addressed the QA problem by extracting useful answers for questions using free text documents instead of restricted knowledge bases in the conventional case.

2.5 Graph Convolutional Networks

Graph representations can effectively capture the characteristics of individuals (as nodes) and their relationships (as edges) in various kinds of problems. Examples of this include social networks, protein structures, and supply chains. Scarselli et al. [149] proposed a graph neural network model that can distill information from an input graph to perform supervised learning tasks. Kipf and Welling [80] proposed the graph convolutional neural networks (GCNs) based on the concept of graph convolution. Inspired by convolutional neural networks (CNNs) [91], GCNs aggregate information from neighbor nodes into node embedding vectors, which is similar to how CNNs extract information from neighbor pixels in images. GCNs have been successfully applied in various domains such as image processing [35, 85], text mining [56], and molecular modelling [38].

Basically, GCNs follow a common design [38] that aims at learning a hypothesis function to generate a representation of node features and their relationships from a graph $G = (\mathrm{V}, \mathrm{E})$. The input for a GCN includes 1) node feature matrix $\mathrm{V} \in \mathbb{R}^{\mathrm{N} \times \mathrm{U}}$, where N is the number of nodes and U is the dimension of the node's feature vector, and 2) the graph adjacency matrix $\mathrm{A} \in \mathbb{R}^{\mathrm{N} \times \mathrm{N}}$ representing the relationships between the nodes. The output of a GCN layer is

calculated through a convolution process [80] as follows:

$$f(H^i, A) = \text{Sigmoid}((\hat{D}^{-\frac{1}{2}} \cdot \hat{A} \cdot \hat{D}^{-\frac{1}{2}} \cdot H^i) \cdot W^i + b^i) \tag{2.25}$$

where H^i is an input node embedding matrix which is V for the first GCN layer or the activation output of the previous layer in case of a hidden GCN layer, $\hat{A} = (A + I)$ is the adjacency matrix added to the identity matrix I, $\hat{D}$ is the diagonal degree matrix of $\hat{A}$, W^i is the weight matrix of the GCN layer, and b^i is the bias vector of the GCN layer. Figure 2.11 shows the graph convolutional network model.

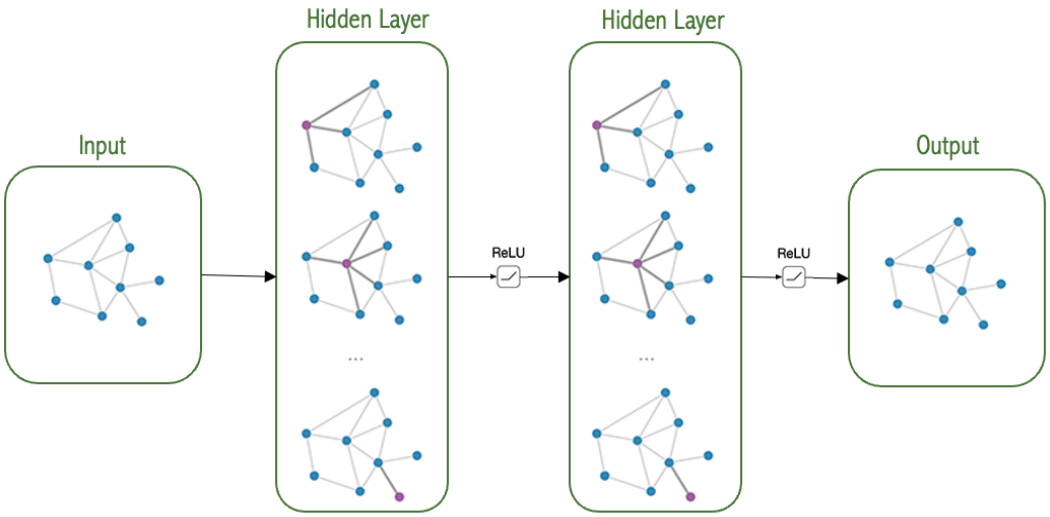


Figure 2.11: Graph Convolutional Network (GCN) model

2.6 Summary

In this chapter, we highlighted the background concepts related to the KT domain. We started with an overview of the intelligent tutoring systems as the main application area for KT models. Then, we introduced the definition for the knowledge tracing problem, and demonstrated it with a high-level example. Next, we presented the learning curve theory as a mathematical foundation to understand the learning and forgetting behaviors of humans impacting knowledge acquisition and retention during the learning process. Finally, we presented methodological concepts for three of the most commonly used modeling approaches in the knowledge tracing literature, including sequence models, memory-augmented models, and graph network models. In the following Chapter, we provide a comprehensive review of the related work in solving the knowledge tracing problem.

Chapter 3

Related Work

This chapter introduces the related work that addresses the KT problem. Section 3.1 defines the knowledge tracing problem with an illustrative example. We then present a categorization of KT methods and discuss the characteristics, limitations, and assumptions of each category in Section 3.2. After that, we describe the datasets used in the experimental evaluations throughout this thesis in Section 3.3. Finally, Section 3.4 concludes the chapter.

3.1 Knowledge Tracing Problem

The *Knowledge Tracing* (KT) problem formulates the challenge of tracing a student's knowledge states based on their exercise answering sequence [137, 3]. Specifically, conditioned on an exercise answering sequence represented as a sequence of question-answer pairs, a knowledge tracing model aims at predicting the likelihood of correctly answering a new question.

Figure 3.1 depicts a probabilistic graphical model for a KT scenario. At each time point t_i in an exercise answering sequence that spans from time t_1 to t_L, we observe a question tag q_i (i.e., question text) and an answer $a_i \in \{0, 1\}$. A student knowledge state at a given time point is represented by a vector over the knowledge states of the involved learning concepts $[C_1, C_2, \ldots, C_N]$, where a learning concept C_n could be a topic in a course such as addition, or subtraction in an elementary math subject. It has to be noted that the KT literature tends to refer to learning concepts as knowledge components (KCs) [3] as they constitute the components of a student's knowledge state.

The KT task involves a hierarchical dynamical system that consists of two levels, including the individual proficiency states of KCs and the overall student's knowledge state. At the top part of Figure 3.1, we show a hidden Markov model representing how an individual learning concept's knowledge state could vary through the learning process, including *known* for a sufficient knowledge, *unknown* for insufficient knowledge, and *forget* for forgetting what has been known before on this concept. We note that knowledge misconception is assumed to be under the *unknown* category as insufficient knowledge that shall result in wrong answers during exercise practice. Early attempts that aimed at addressing the KT problem are dated back to three decades ago, where Bayesian inference methods were used to predict the likelihood of correctly answering a new question given the exercise answering sequence [31]. Recently, deep learning approaches [137, 193, 118] have shown promising potential in addressing the KT problem due to the representation capacity of deep neural networks and the use of advanced sequence

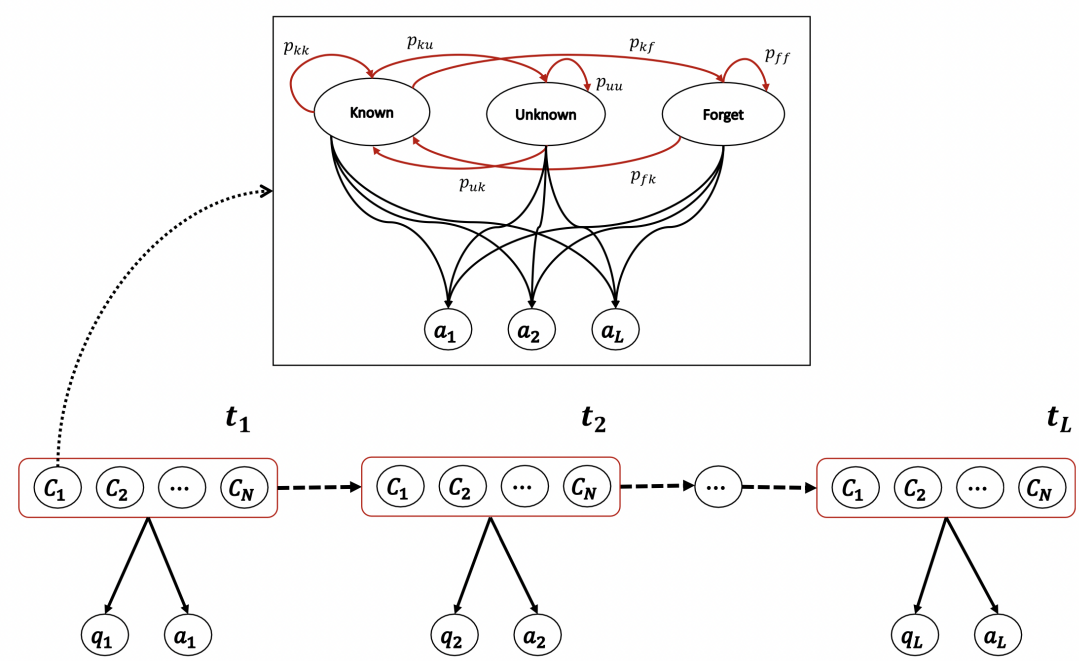


Figure 3.1: A probabilistic graphical model representing a knowledge tracing scenario.

modeling techniques such as differential associative memory [201, 1] to capture long-term dynamics in the exercise answering sequence, graph neural networks [173, 195, 122, 2] for incorporating relationships among learning concepts and questions, and attention [127, 126, 45, 26] for focusing on relevant questions in the exercise answering sequence. The performance of deep neural models is significantly impacted by the volume and quality of available training data. Thus, there is a need for datasets that can reflect realistic dynamics in an online learning process.

3.2 Categorization Of Knowledge Tracing Models

This section introduces a comprehensive categorization of the KT models according to the related works found in the literature. Generally speaking, there are two broad categories: (1) *Traditional Knowledge Tracing Models*; and (2) *Deep Learning Knowledge Tracing Models*. We review methods from these two categories and present their different application assumptions as follows.

3.2.1 Traditional Knowledge Tracing Models

Traditionally, there are two popular lines of research for knowledge tracing: the *Bayesian Knowledge Tracing* and the *Factor Analysis Models*. Figure 3.2 provides an overview of major traditional knowledge tracing models that have been developed in the KT literature.

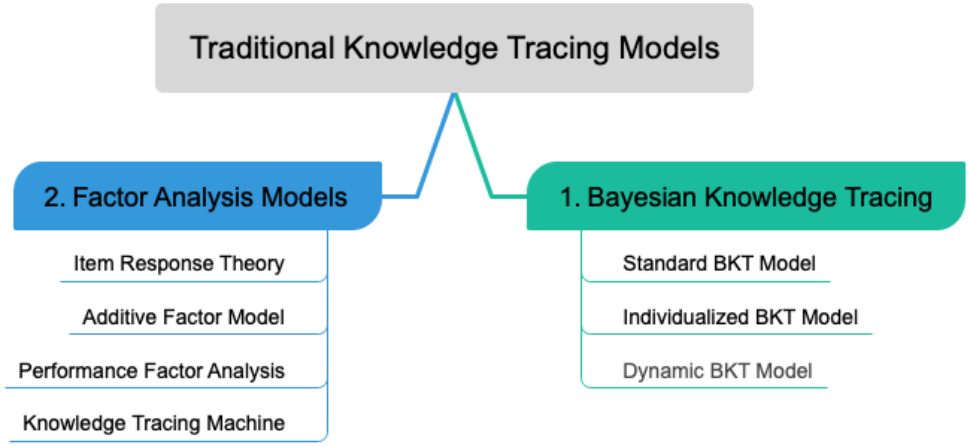


Figure 3.2: An overview of traditional knowledge tracing models.

3.2.1.1 Bayesian Knowledge Tracing

Bayesian Knowledge Tracing (BKT) was motivated by the concepts of *mastery learning* [30]. Mastery learning assumes that all students can practice a skill such that it may lead to mastery of that skill if two conditions are satisfied: (a) knowledge is appropriately described as a hierarchy of skills; and, (b) learning experiences are structured to ensure that students master skills lower than those higher in the hierarchy [31].

BKT models often use a probabilistic graphical model such as Hidden Markov Model [33] and Bayesian Belief Network [179] to trace students' changing knowledge states as they practice skills. Central to these models is the Bayes' theorem that, for two events A and B, the following holds:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}. \tag{3.1}$$

In what follows we discuss the standard Bayesian approaches and their variations.

- **Standard BKT Model**

  The first BKT model was introduced by Corbett and Anderson in 1994 [31]. The proposed model associates a skill with a binary knowledge state: $\{unlearned, learned\}$. This model only considers transitions from the unlearned state to the learned state, overlooking the forgetting theory (i.e., the probability of transition from a learned state to an unlearned state is always zero). Additionally, note that a student may make a mistake while in a learned state or guess correctly in an unlearned state. We refer to this model as the *standard BKT model* from now on. Table 3.1 summarizes the main parameters of the BKT model.

  There are two types of variables in the standard BKT model: (1) the *Binary latent variables* which represent the knowledge states of a given student (i.e., a single variable

Parameter	Description
$p(L_0)$	Probability of skill mastery by a student before learning
$p(T)$	Probability of transition from an unlearned state to a learned state
$p(S)$	Probability of slipping by a student in a learned state
$p(G)$	Probability of guessing correctly by a student in an unlearned state

Table 3.1: Four types of model parameters used in BKT

per skill indicating *learned state* and *unlearned state*); and, (2) the *Binary observed variables* which represent how students attempt questions (i.e., a variable per question indicating whether a question is answered correctly or not).

Figure 3.3 shows four types of model parameters in the standard BKT model. For each skill, there is one set of four corresponding parameters. At each time step $n \geq 1$, the model estimates the probability $p(L_n)$ of skill mastery by a student by:

$$p(L_n) = \text{Posterior}(L_{n-1}) + (1 - \text{Posterior}(L_{n-1})) * p(T), \tag{3.2}$$

where Posterior (L_{n-1}) is the posterior probability of being in a learned state given the observation to the n-th attempt by a student, calculated as:

$$\text{Posterior}(L_{n-1}) = \begin{cases} \dfrac{p(L_{n-1}) * (1 - p(S))}{p(L_{n-1}) * (1 - p(S)) + (1 - p(L_{n-1})) * p(G)} & \text{if the n-th attempt is } \textit{correct}; \\ \dfrac{p(L_{n-1}) * p(S)}{p(L_{n-1}) * p(S) + (1 - p(L_{n-1})) * (1 - p(G))} & \text{otherwise.} \end{cases}$$

Hence, the probability of a student to correctly answer a question at each time step *n* is the sum of the probability of either mastering the skill but making a "slip" or not mastering the skill but making a correct "guess", as formulated by:

$$p(L_n) * (1 - p(S)) + (1 - p(L_n)) * p(G)). \tag{3.3}$$

Figure 3.3a shows the standard BKT model with one skill node k. Starting with the prior probability $p(L_0)$ of skill mastery, the latent variable for skill k is transitioned from one time step $t - 1$ to the next time step t based on the probability $p(T)$. The corresponding observed variable y represents the answer node, i.e., how a student attempts questions that require skill k, which is based on the probabilities $p(G)$ and $p(S)$. Table 3.1 summarizes the main parameters for the BKT model.

- **Individualized BKT Model**

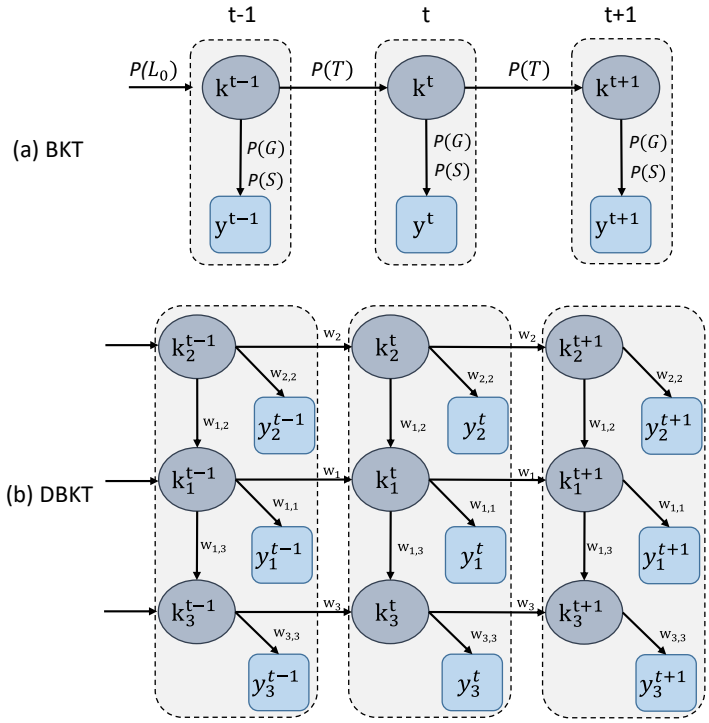


Figure 3.3: Comparing the model architectures between (a) BKT and (b) DBKT, where latent variables for skills at the time step t are denoted as k_i^t and their correspondingly observed variables for answers at the time step t are denoted as y_i^t.

One limitation of the standard BKT model is that it has no model parameters specific to students. All students are assumed to have the same prior knowledge and the same learning rate for any skill. As a result, the standard BKT model may underestimate the learning performance of above-average students, but overestimate the learning performance of below-average students [31].

To alleviate this limitation, several attempts [31, 131, 93, 199, 78] have been made to extend the standard BKT model by introducing student-specific parameters. Corbett and Anderson [31] considered to add individual weights for each student, one weight corresponding to each of the four-parameter types in the standard BKT model. Pardos and Heffernan [131] focused on individualizing the prior probability of skill mastery $p(L_0)$ heuristically for each student. Lee and Brunskill [93] also individualized the four-parameter types in the standard BKT model for students; nonetheless, the combined effects of skill-specific and student-specific parameters were unexplored. Yudelson et al. [199] introduced an approach for individualizing BKT models that account for student differences with respect to two types of model parameters, the prior probability of skill mastery $p(L_0)$ and the probability of transition $p(T)$. The idea was to first define student-specific parameters and skill-specific parameters. Then, the gradients were explicitly

Parameter	[31]		[131]		[93]		[199]		[78]	
	skill	student	skill	student	skill	student	skill	student	skill	student
$p(L_0)$	✓	✓	✓	✓	-	✓	✓	✓	✓	-
$p(T)$	✓	✓	✓	-	-	✓	✓	✓	✓	-
$p(S)$	✓	✓	✓	-	-	✓	✓	-	✓	✓
$p(G)$	✓	✓	✓	-	-	✓	✓	-	✓	✓

Table 3.2: Model parameters of individualized BKT models

computed in terms of both student-specific and skill-specific parameters. The underlying Hidden Markov Model remains unchanged. It turns out that adding a student-specific parameter for $p(T)$ is more beneficial for the model accuracy than adding a student-specific parameter for $p(L_0)$. Later, Khajah et al. [78] proposed to extend the standard BKT model by personalizing the guess and slip probabilities $p(G)$ and $p(S)$ based on student ability and problem difficulty.

Compared to the standard BKT model, the individualized models can provide a better correlation between actual and expected accuracy among students, leading to more effective decisions or improving the accuracy of predicting student performance. Table 3.2 summarizes the skill- and student-specific parameters used in the individualized BKT models.

- **Dynamic BKT Model** In the early years, the BKT models have assumed that each question requires only one skill and different skills are independent of each other [31, 131, 93, 199]. Thus, these models cannot handle questions that require multiple skills, nor represent relationships between different skills. To address this limitation, Käser et al. [76] proposed to jointly model multiple skills and dependencies between different skills using *Dynamic Bayesian Network* (DBN). They aimed to capture prerequisite skill hierarchies within a single model, e.g., one skill is conditionally dependent on another skill if the former is a prerequisite for mastering the latter. Similar to the standard BKT model, DBN considers the same two types of variables: binary latent variables and binary observed variables.

  At each time step, a latent variable for each skill is associated with an observed variable. A forget probability $p(F)$ is introduced, in addition to the four types of model parameters $\{p(L_0), p(T), p(S), p(G)\}$ in the standard BKT model. Dependencies between different skills are learned as weights $\mathbf{w}$ of a log-linear model. Let $f : X \times O \rightarrow \mathbb{R}^d$ denote a mapping from a latent space X and an observed space O to d-dimensional feature vectors, and c be a normalizing constant. The objective of the log-linear model is to find the model parameters $\{p(L_0), p(T), p(S), p(G), p(F), \mathbf{w}\}$ that maximize the likelihood of the joint probability of $x_i \in X$ and $y_i \in O$ as formulated below:

$$L(\mathbf{w}) = \sum_i \ln\Big(\sum_{x_i} \exp(\mathbf{w}^\intercal f(x_i, y_i) - c)\Big).$$

Figure 3.3b shows the Dynamic Bayesian Network (DBKT) with three skill nodes (denoted as k_1, k_2, and k_3). As indicated by the directed arrows, the latent variable for skill k_2 depends on the latent variable for skill k_1, and the latent variable for skill k_1 depends on the latent variable for skill k_3. Further, at each time step t, latent variables for skills depend on their latent variables in the previous time step $t-1$, while y_i is the corresponding observed answer node.

3.2.1.2 Factor Analysis Models

Factor analysis models are theoretically supported by the *Item Response Theory* (IRT) [40], which has played a large role in educational assessment and measurement. The key idea is to estimate student performance by learning a function, usually a logistic function, based on various factors in a population of students who solve a set of problems. It is important to note that, although an item in the original IRT corresponds to a question involving a single skill, later works in this line have been generalized to consider an item that may involve multiple skills. The mapping between items and skills is often represented in the form of a Q-matrix, i.e., an entry q_{jk} in a Q-matrix is 1 if an item j involves a skill k; or 0, otherwise. Q-matrix is commonly assumed as the side information.

- **Item Response Theory (IRT)**

  The history of IRT can be traced back to Thurston's pioneering work in the 1920s [108] and several other works in the 1950s and 1960s [54, 16, 6, 109]. IRT is built upon the following assumptions: (a) The probability that a student correctly answers an item can be formulated as an *item response function* based on the parameters of the student and the item; (b) The item response function monotonically increases with respect to the ability θ_i of a student i; (c) For a student with the ability θ_i, items are considered conditionally independent.

  The *Rasch* model [141] is often referred to as the simplest IRT model, in which the item response function is defined by a one-parameter logistic regression (1PL) model. Let $L(\cdot)$ be a logistic function. By taking into account a difficulty parameter b_j that models the difficulty of an item j, the probability p_{ij} that a student i correctly answers an item j is defined as:

  $$p_{ij} = L\Big(\theta_i - b_j\Big) = \frac{\exp^{(\theta_i - b_j)}}{1 + \exp^{(\theta_i - b_j)}}. \tag{3.4}$$

  Several multiple-parameter logistic regression models have also been developed for IRT, e.g., a four-parameter logistic (4PL) model introduced by Barton and Lord [9]:

  $$p_{ij} = c_j + (d_j - c_j)L\Big(a_j(\theta_i - b_j)\Big) = c_j + (d_j - c_j)\frac{\exp^{a_j(\theta_i - b_j)}}{1 + \exp^{a_j(\theta_i - b_j)}}, \tag{3.5}$$

  where a_j is a discrimination parameter to model how well an item j can differentiate students, c_j is a guessing parameter to model the effect of guessing, and d_j is a slipping parameter to model the effect of careless errors.

IRT has been extended in many different directions. Wilson et al. [190] proposed *Hierarchical IRT* (HIRT) and *Temporal IRT* (TIRT). HIRT exploits structure among questions by assuming that related questions (i.e., questions sharing similar skills) have difficulty parameters drawn from the same distribution. Different questions might vary in difficulty, but questions for trivial skills tend to be easier while ones for difficult skills tend to be harder. TIRT models each parameter in the logistic model (e.g., 4PL) as a time-varying stochastic process such as a *Wiener random* process [167]. Zhou et al. [202] proposed the *Educational context-aware Cognitive Diagnosis (ECD)* model that builds on the IRT theory by combining a student's educational context features (e.g., gender or parents' educational level) with exercise answering features using a two-stage hierarchical attentive architecture. Zhuang et al. [204] proposed a question recommendation framework for enhancing a student's proficiency level, named *Neural Computerized Adaptive Testing (NCAT)*, which consists of two components: a question selection agent following a Q-learning setup [187] and an IRT-based knowledge tracing model [182] for answer prediction. The NCAT framework firstly selects a support set (i.e., a train mini-batch of question-answer samples) that is used to train the IRT-based knowledge tracing model, then uses a query set (i.e., a test mini-batch) to evaluate the performance of the model, and generates the reward signal for the question selection agent as the inverse of the prediction error on the query set.

- **Additive Factor Model (AFM)**

  The *Additive Factor Model* [21], originated from *Learning Factors Analysis* (LFA) [20], is a logistic regression model under four assumptions: (1) the prior knowledge of students may vary; (2) students learn at the same rate; (3) some skills are more likely to be known than others; and, (4) some skills are easier to be learned than others. In this model, a difficulty parameter β_k and a learning rate parameter γ_k are assigned for each skill k, respectively.

  The key idea of AFM is that the probability of answering an item correctly by a student is proportional to an additive combination of the ability of the student, the difficulty of skills involved in the item, and the amount of learning gained from each attempt. Let $K(j)$ be the set of skills involved in an item j, which can be obtained from a Q-matrix, and T_{ik} be the number of times that a student i has attempted on an item involving a skill k. AFM defines the probability of answering an item correctly by a student i on an item j as:

$$p_{ij} = L\Big(\theta_i + \sum_{k \in K(j)} (\beta_k + \gamma_k T_{ik})\Big). \tag{3.6}$$

- **Performance Factor Analysis (PFA)**

  Performance Factor Analysis (PFA) [134] overcomes the limitation of AFM which ignores the evidence of learning in the successful and unsuccessful attempts on items by a student.

The key idea of PFA is to discard the parameter θ_i used in the previous models and instead count for successful and unsuccessful attempts separately. PFA has three parameters for each skill, including: (1) β_k is for the difficulty of a skill k; (2) γ_k^S is for the effect of learning a skill k after successful attempts; and, (3) γ_k^F is for the effect of learning a skill k after unsuccessful attempts. Conceptually, γ_k^S and γ_k^F reflect the learning rate for a skill k when being applied successfully and unsuccessfully.

Let T_{ik}^S and T_{ik}^F be the number of successful attempts and the number of unsuccessful attempts made by a student i on a skill k, respectively. Then PFA calculates the probability that a student i correctly answer an item j as:

$$p_{ij} = L\Big(\sum_{k \in K(j)} (\beta_k + \gamma_k^S T_{ik}^S + \gamma_k^T T_{ik}^F) \Big). \tag{3.7}$$

- **Knowledge Tracing Machine (KTM)**

  Knowledge Tracing Machine (KTM) was recently proposed by Vie and Hisashi [178], which generalizes *Factorization Machine* (FM) [169, 166] for student modeling.

  KTM allows considering an arbitrary number of factors about students, items, skills, successful and unsuccessful attempts, or extra information about the learning environment, such as using a mobile or a laptop. For a number N of factors, we denote all factors involved in an event by a sparse vector x of length N such that $x_k > 0$ if a factor $k \in [1, N]$ is involved in the event; or 0, otherwise. Then KTM estimates the probability p_{ij} of a correct answer on an item j by a student i with an event involving x as follows:

$$p_{ij} = L\Big(\sum_{k=1}^{N} w_k x_k + \sum_{1 \leqslant k < l \leqslant N} x_k x_l \langle v_k, v_l \rangle + \mu \Big) \tag{3.8}$$

  where μ is a global bias, and each factor k is modeled by both a weight $w_k \in \mathbb{R}$ and an embedding vector $v_k \in \mathbb{R}^d$ for some dimension d. The first term models the logistic regression of all factors and the second term models pairwise interactions between different factors. When L is a logistic function, KTM includes IRT, AFM, and PFA as special cases [178].

  Wenbin et al. [44] extended the KTM model through their *Knowledge Tracing Machine by modeling cognitive item Difficulty and Learning and Forgetting (KTM-DLF)* model by adding factors related to the forgetting behavior of students. They represented the forgetting by time lapse since the last successful attempt for the involved skills.

3.2.1.3 Discussion

Both Bayesian knowledge tracing and factor analysis models have strengths and weaknesses. We discuss their connections and differences from three aspects: model parameters, model inference, and temporal analysis.

- **Model parameters:** Most recent models in BKT and FAM have taken into account both student- and skill-specific model parameters. Early BKT models were primarily centered on the four parameters: prior learning parameter $p(L_0)$, learning rate parameter $p(T)$, guess parameter $p(G)$, and slip parameter $p(S)$ [31] and their student-specific variants. These early BKT models usually assume that there is no forgetting parameter. Only some recent works have incorporated the forgetting parameter [76]. For factor analysis models, most of them have been developed with similar or more flexible model parameters than BKT models. Particularly, recent works such as KTM [178] consider a wide range of factors, enabling a flexible way to incorporate the side information into student modeling.

- **Model inference:** To keep the Bayesian inference tractable, BKT models typically assume a first-order Markov chain when making inference based on a sequence of past question-answering history, i.e., only considering the most recent observation. This assumption however limits their ability to model complex dynamics on student learning behaviors. Some recent dynamic BKT models such as DBN [76] are often computationally intractable, and thus they trade-off prediction accuracy for computational efficiency. On the other hand, factor analysis models usually do not explicitly make inferences on the knowledge states of a student (e.g., decide whether a student has achieved a certain level of skill mastery by tracing knowledge states). Instead, they target to maximize other model parameters such as the learning rate.

- **Temporal analysis:** BKT models essentially deal with a sequence prediction problem based on the history of student learning. In contrast, factor analysis models do not consider the order of questions in which a student's answers are observed. For example, given two questions and their corresponding answers from a student, whether one question is answered before the other is not important for factor analysis models. Nonetheless, by incorporating extra temporal features of student learning behaviors, factor analysis models can be enhanced to analyze temporal aspects of student learning.

A number of attempts have been made to leverage the best from both worlds of Bayesian knowledge tracing and factor analysis models. For example, [66, 77, 78] extended the IRT using the Bayesian inference to customize the estimation of question difficulty based on observations of each student. Further work has been done by incorporating factors that reflect the characteristics of individual students [33, 131], or the characteristics of specific items of assessment within skills [132].

3.2.2 Deep Learning Knowledge Tracing Models

Inspired by the success of deep learning [151, 163, 51, 48, 92, 183], recent research on knowledge tracing has applied deep learning techniques. Figure 3.4 presents a taxonomy of deep learning knowledge tracing models.

Figure 3.4: A taxonomy of deep learning knowledge tracing models.

3.2.2.1 Sequence Modeling for Knowledge Tracing

The knowledge tracing task can be represented through a sequence prediction objective from a machine learning perspective. That is given the previous sequence of question answering. We need to predict the probability of answering the current question correctly (see Section 1.1.2). Following this representation, it is necessary to introduce sequence modeling methods that appear in the knowledge tracing literature.

- **Deep Knowledge Tracing (DKT)**

  Deep Knowledge Tracing (DKT) [137] pioneered the use of deep learning for knowledge tracing. It employs *Recurrent Neural Network* (RNN) [98] and a Long Short Term Memory (LSTM) [64] to predicate the probability of correctly answering a question at each time step. A sequence of hidden states $\langle h_1, h_2, \ldots, h_n \rangle$ is computed which encodes the sequence information obtained from previous interactions. At each time step t, the model calculates the hidden state h_t and the student's response p_t as follows:

$$h_t = \text{Tanh}(\mathbf{W}_{hx} x_t + \mathbf{W}_{hh} h_{t-1} + b_h) \tag{3.9}$$

$$p_t = \sigma(\mathbf{W}_{hy} h_t + b_p) \tag{3.10}$$

where the $\text{Tanh}(u_i) = (e^{u_i} - e^{-u_i})/(e^{u_i} + e^{-u_i})$ and $\sigma(u_i) = 1/(1 + e^{-u_i})$ are activation functions, $\mathbf{W}_{hx}$, $\mathbf{W}_{hh}$ and $\mathbf{W}_{hy}$ are weight matrices, and b_h and b_p are bias vectors.

Despite the promising performance, DKT has several limitations. First, it assumes only one hidden KC (i.e., a skill) in a student's knowledge state h_t. Second, it cannot model the relationships among multiple KCs. Third, it assumes that all questions are equally likely related to each other, which may not hold in many scenarios as some questions may be more relevant to each other than the remaining of the sequence. Thus, various attempts have been made on extending DKT with the aim of enhancing the model capacity for tackling the KT problem. Below, we review the related work in the following areas of extension.

A number of KT models have extended DKT [137] to address its limitations. For example, Xiong et al. [193] proposed *Extended-Deep Knowledge Tracing* that extends DKT by adding auxiliary student features such as previous knowledge, question answering rates, and time spent on learning and practice; and, exercise features, such as textual information, question difficulty, skill hierarchies, and skill dependencies. A variant of DKT, called (DKT+) [197], was proposed to augment the original DKT loss function with two additional regularization terms to address the limitations in DKT's ability to reconstruct an answer input and reduce inconsistency of answer prediction for questions sharing similar KCs. Minn et al. [115] proposed an extension of DKT, named *Deep Knowledge Tracing with Dynamic Student Classification* (DKT-DSC), that uses K-means to cluster student profiles into groups based on their performance over the KCs and dynamically update the current cluster information over time while the performance changes.

3.2.2.2 Memory-Augmented Knowledge Tracing Models

To trace complex KCs learned by students, several works have extended DKT by augmenting an external memory structure, inspired by memory-augmented neural networks [52]. In particular, following *Key-Value Memory Network* (KVMN) [114], a key-value memory has been employed to represent the knowledge state, which has more representational power than a hidden variable used in DKT. Such a key-value memory consists of two matrices: *key* and *value*. The *key* matrix stores the representations of KCs and the *value* matrix stores the student's mastery level of each KC. Below, we discuss two popular key-value memory networks for knowledge tracing.

- **Dynamic Key-Value Memory Network (DKVMN)**

  Dynamic Key-Value Memory Network (DKVMN) [201] has augmented DKT with two memory matrices: *key* and *value*. To trace how the knowledge state of a student evolves over time, unlike KVMN in which both *key* and *value* matrices are static [114], DKVMN designs the *value* matrix to be dynamic while remaining the *key* matrix to be static.

  Figure 3.5a shows the model architecture of DKVMN, where $\mathbf{M}^k \in \mathbb{R}^{N \times d_k}$ is the *key* matrix and $\mathbf{M}_t^v \in \mathbb{R}^{N \times d_v}$ is the *value* matrix at the time step t. It is assumed that there are

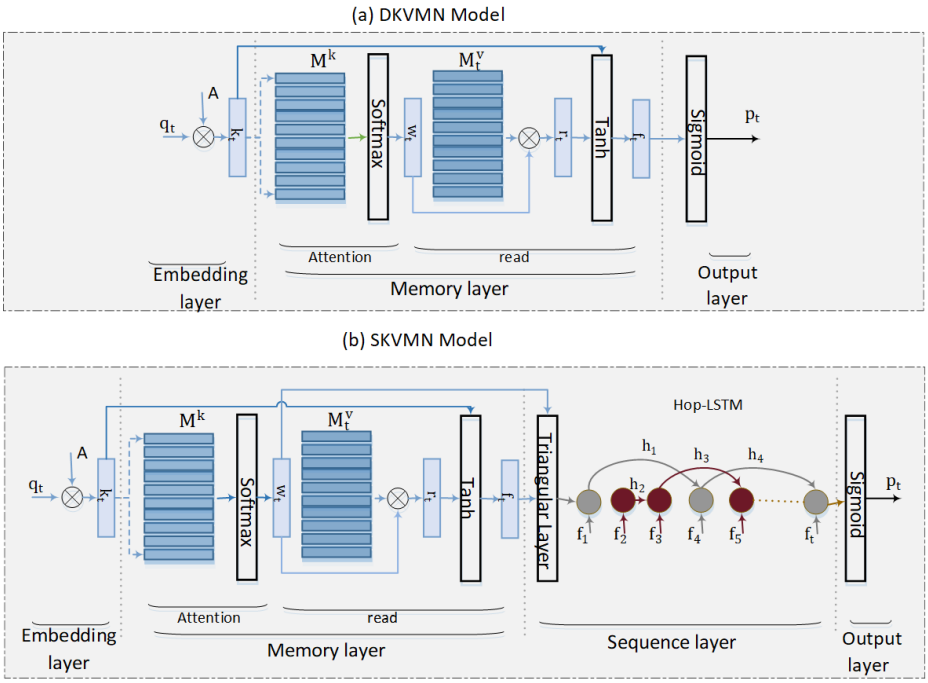


Figure 3.5: A comparison of the model architecture design between (a) DKVMN and (b) SKVMN.

N latent KCs underlying all questions in a learning task. For a question q_t at the time step t, a correlation weight w_t is computed, which represents the correlation between the question q_t and the underlying latent KCs stored in the *key* matrix $\mathbf{M}^k$. The model first retrieves a student's knowledge state with regard to the question q_t from the *value* matrix $\mathbf{M}_t^v$, calculated as:

$$r_t = \sum_{i=1}^{N} w_t(i)\mathbf{M}_t^v(i). \tag{3.11}$$

Then, the student's response to the question q_t is predicted based on the retrieved knowledge state. After the student answers the question q_t, the *value* matrix is updated to reflect the knowledge growth of the student after working on q_t.

- **Sequential Key-Value Memory Network (SKVMN)**

  Sequential Key-Value Memory Network (SKVMN) [1] aimed to address a limitation in DKT and DKVMN that KCs required by answering the past questions in a sequence are not necessarily relevant to KCs required by answering the current question. Thus, SKVMN employs a modified LSTM for sequential modeling, called *Hop-LSTM*, while remaining the same key-value memory structure and loss function as in DKVMN. Different from the standard LSTM, Hop-LSTM can explicitly capture sequential dependencies among questions in a sequence of interactions, and update the knowledge state of a student based on their responses to relevant questions.

  Figure 3.5b shows the model architecture of SKVMN. More precisely, two LSTM cells

in Hop-LSTM are connected only if the input question of one LSTM cell is sequentially dependent on the input question of the other LSTM cell. This means that Hop-LSTM has the capability of hopping across the LSTM cells when their input questions are irrelevant to the current question q_t. Thus, it enables to capture long-term dependencies among questions that require similar KCs.

3.2.3 Attentive Knowledge Tracing Models

Following the *Transformer* architecture [177], several works [126, 45, 26, 157, 127] have attempted to incorporate an attention mechanism into KT models. Although the attention mechanisms introduced by these works vary, their key ideas are similar, i.e., to learn the attention weights of questions in a sequence of interactions in a way that can reflect the relative importance of these questions for predicting the probability of correctly answering the next question. This mitigates one limitation of DKT which treats all questions in a sequence of interactions as equally important. In what follows, we discuss the main attentive knowledge tracing models. Table 3.3 briefly summarizes these models.

KT Model	Forgetting-Aware	Text-Aware	Attention Mechanism
SAKT [126]	×	×	Multi-head self-attention
AKT [45]	✓	×	Monotonic attention
SAINT [26]	×	×	Multi-head self-attention
SAINT+ [157]	✓	×	Multi-head self-attention
RKT [127]	✓	✓	Relational multi-head self-attention
CKT [156]	×	×	Dot-product attention
CoKT [106]	×	×	Collaborative multi-head self-attention

Table 3.3: A comparison of attentive knowledge tracing models.

- **Self-Attentive Knowledge Tracing (SAKT)**

  Self-Attentive Knowledge Tracing (SAKT) [126] was the first to add an attention mechanism into the KT models. It uses the scaled dot-product attention mechanism proposed by Vaswani et al. [177] to learn attention matrices using multiple attention heads. Specifically, each attention matrix contains relative weights from a representative subspace, which indicates the importance of questions in past interactions for predicting a student's answer to the current question. Then, attention matrices from different representative subspaces are sent to a feed-forward network for predicting student performance.

- **Attentive Knowledge Tracing (AKT)**

  Attentive Knowledge Tracing (AKT) was proposed by Ghosh, Heffernan, and Lan [45]. AKT differs from SAKT in its attention mechanism called *monotonic attention* (i.e., a modified, monotonic version of the scaled dot-product attention mechanism [177]) that can reduce attention weights for questions in a sequence of interactions proportional to their time distance in an exponential decay rate. The exponential weight decay is meant to consider the forgetting effect in a student's memory over time. In addition, an

embedding representation was proposed to take into account a parameter for controlling how far a question deviates from a knowledge component it involves by following the *Rasch* model [141].

- **Separated Self-AttentIve Neural Knowledge Tracing (SAINT)**

  Separated Self-AttentIve Neural Knowledge Tracing (SAINT), proposed by Choi et al. [26], differs from AKT and SAKT in the way that it has further applied an encoder-decoder model along with the scaled dot-product attention mechanism as in the original architecture of Transformer [177]. Specifically, SAINT separates a sequence of interactions by a student into a *question embedding sequence* and a *response embedding sequence*, which are then sent to the encoder and the decoder as input, respectively. The encoder and decoder are combinations of multi-head attention networks with the scaled dot-product attention mechanism [177].

  Recently, SAINT was extended by adding two time-related features into a response embedding sequence: *elapsed time* for the time taken by a student to answer each question, and *lag time* for the time interval between two consecutive learning interactions. This variant was named the SAINT+ model [157].

- **Relation-Aware Self-Attention for Knowledge Tracing (RKT)**

  Relation-Aware Self-Attention for Knowledge Tracing (RKT) was proposed by Pandey and Srivastava [127]. Similar to SAKT and SAINT, RKT employs the scaled dot-product attention mechanism proposed by Vaswani et al. [177] to learn attention weights using multiple attention heads. However, as differs from the other attention-based KT models, RKT combines attention weights with *relation coefficients*, which are obtained from *exercise relation modeling* and *forgetting behavior modeling*. For the exercise relation modeling, it leverages the text information of questions (e.g., a question's textual information) to represent questions and estimate the relation between questions in a sequence of past interactions. For forgetting behavior modeling, similar to AKT, RKT considers an exponential decay to count for a student's forgetting behavior over time.

- **Convolutional Knowledge Tracing (CKT)**

  Shen et al. [156] proposed the CKT model that combines attention with 1-D convolutional networks [81] for predicting correct answers. In addition to the past question-answering sequence, the CKT model considers a student's individualized skills represented as the historically relevant performance and the overall concept performance. The former uses a masked-attention between the latest question and each previous question in the answer sequence to extract a mastery score, while the latter extracts the mastery level on the learning concepts level by getting the percentage of the correctly answered questions for each defined concept. During answer prediction, the CKT combines individualized features with embeddings from the answer sequence using a linear gating mechanism and applies 1-D convolution operations to extract the final input representation for answer prediction.

- **Collaborative Knowledge Tracing (CoKT)**

  Collaborative Knowledge Tracing (CoKT) [106] borrowed the idea of collaborative filtering from the recommendation literature in order to consider the knowledge states of peer students in predicting the answer probability of a given student. The authors empirically showed that fusing the intra-state (extracted from the previous answer history) and inter-state (extracted from knowledge states from peer students) features could enhance the answer prediction performance for students with a short answer history. The model estimates a peer's similarity score by deploying the *BM25* [144] string similarity function between the string-encoded question answering sequences and projects embedding vectors representing the inter-state from similar peers using a multi-head self-attention mechanism [177]. Similarly, the previous answer history of a student is embedded using a multi-head self-attention to represent the intra-state and is combined with the intra-state to get the final answer prediction.

3.2.3.1 Graph-Based Knowledge Tracing Models

In KT tasks, various relational structures often exist, for example, the similarity of KCs, the dependency between KCs, and the correspondence between questions and their KCs. To capture such relational structures for better addressing the KT problem, a recent trend is to explore the power of graph representation learning techniques such as Graph Neural Networks (GNN). Table 3.4 briefly summarizes several main graph-based knowledge tracing models and we discuss each of them separately.

KT Model	Graph		Assumption
	Node	Edge	
GKT [122]	KC	KC-KC relation	One KC per question
GIKT [195]	Question or KC	Question-KC relation	Many KC per question
SKT [173]	KC	KC-KC multiple relations	One KC per question
PEBG [105]	Question or KC	Question-KC relation	Many KC per question

Table 3.4: A comparison of graph-based knowledge tracing models.

- **Graph-based Knowledge Tracing (GKT)**

  Graph-based Knowledge Tracing (GKT), proposed by Nakagawa, Iwasawa, and Matsuo [122], attempted to incorporate a graph where nodes represent KCs and edges represent the dependency relation between KCs for a relational inductive bias. They reformulated the KT problem as a time series node-level classification problem and solved it using standard graph learning techniques such as message-passing GNNs [149]. Since such a graph is not explicitly given in KT tasks, the authors proposed two approaches to construct such graphs from a sequence of interactions by a student: (1) a Statistics-based approach to construct a graph based on statistics such as how many times one KC was answered after another KC was answered; (2) Learning-based approach to learn a graph with performance optimization in an end-to-end manner.

- **Graph-based Interaction Knowledge Tracing (GIKT)**

  Graph-based Interaction Knowledge Tracing (GIKT), proposed by Yang et al. [195], leverages the relationship between questions and KCs, represented as a graph to learn useful embedding for answer prediction. Different from GKT which implicitly assumes that each question corresponds to one KC, GIKT assumes that one KC may be related to many questions and one question may correspond to more than one KC. Thus, GIKT can use a GNN to aggregate the embeddings of questions and KCs based on their relationship in the graph and sends the embedding of each question in a sequence of interactions to an RNN model to predict a student's answer for the next question.

- **Structure-based Knowledge Tracing (SKT)**

  Structure-based Knowledge Tracing (SKT) was proposed by Tong et al. [173] which aimed to capture multiple relations among KCs such as similarity relation and prerequisites relation. Similar to GKT, SKT also assumes that each question corresponds to one KC. However, instead of a single relation between KCs as captured in GKT, SKT exploits multiple relations between KCs. Further, SKT supports information propagation to jointly model the temporal and spatial effects when summarizing graph data. These two kinds of graph embeddings are combined at each time step and fed to a recurrent model to predict the correct answer by a student.

- **Pre-training Embeddings via Bipartite Graph (PEBG)**

  Pre-training Embeddings via Bipartite Graph (PEBG) [105] presented a method for obtaining pre-trained exercise embeddings so as to improve the accuracy of knowledge tracing. In KT tasks, explicit exercise-KC relations and implicit exercise similarity as well as KC similarity often exist simultaneously. To capture all these relations in exercise embeddings, the authors represent them together with exercise difficulties as a bipartite graph. Then, they fuse these features in the defined bipartite graph to obtain the pre-trained exercise embeddings. Their experiments have indicated that the obtained exercise embeddings can significantly improve the performance of some KT models, such as DKT [137].

3.2.3.2 Text-Aware Knowledge Tracing Models

Until now, the deep KT models that we have discussed mainly focused on a student's interactions with questions in a sequence to predict the probability of correctly answering the latest question by the student. Yet, they did not consider the textual features of the questions themselves. Text-aware KT models are motivated by leveraging the textual features of questions to enhance performance in tackling KT tasks.

- **Exercise-Enhanced Recurrent Neural Network (EERNN)**

  Exercise-Enhanced Recurrent Neural Network (EERNN) was proposed by Su et al. [161], which is a text-aware KT model to predict the probability of correctly answering a

given question. The model uses a bi-directional LSTM module to extract the representation (i.e., a vector) of each question from the question's text and then trace a student's knowledge states by combining it with the representations of the previously answered questions using another LSTM module. Two variants of EERNN were developed: EERNNM, and EERNNA. The EERNNM variant assumes that a sequence of interactions satisfies the Markov property, i.e., the answer prediction for the next question only depends on the latest observed knowledge state; thus it only considers the last hidden state. The EERNNA variant considers all the previous knowledge states and combines them through an attention mechanism.

Later, Yin et al. [198] further extended the work by Su et al. [161] by leveraging a pre-training task to learn question representations. The authors followed a masked language model (MLM) objective [112] and showed that this pre-training step could further enhance the model's performance compared to the original model.

- **Exercise-Aware Knowledge Tracing (EKT)**

  Exercise-Aware Knowledge Tracing (EKT) [101] extends EERNN to incorporate the information of multiple KCs during answer prediction, where a student's knowledge state is represented by a knowledge state matrix, rather than a knowledge state vector. Specifically, the model uses a memory network for quantifying how much each question can affect the mastery of a student on multiple KCs during a sequence of interactions by the student.

- **Adaptable Knowledge Tracing (AdaptKT)**

  Cheng et al. [25] addressed the problem of transfer learning across KT domains aiming to transfer a trained KT model on the source domain to operate on a target domain while preserving its performance. Their proposed method works on the textual data of questions through learning a question embedding using a deep auto-encoder architecture that is trained on questions from both domains and then training a KT model on the source domain while regularizing it to minimize the mean discrepancy [171] between the knowledge states in both domains. The final layer is randomly reinitialized while keeping the weights of the earlier layers frozen to train on the target domain.

In addition to the above text-aware KT models, other types of KT models such as *Relation-Aware Self-Attention for Knowledge Tracing* (RKT) [127] and *Hierarchical Graph Knowledge Tracing* (HGKT) [172] also extract features from the textual information of questions for learning question representations in their models.

3.2.3.3 Forgetting-Aware Knowledge Tracing Models

Learning psychological studies [74, 135, 153] showed that forgetting is an important aspect to consider for an accurate estimation of a student's knowledge state. This is because the knowledge mastery level of a student tends to decline at an exponential rate over time since the last practice of the relevant questions. From an experimental psychology perspective, Hermann

Ebbinghaus [39, 120] studied forces that affect memory retention, leading to formulate what is currently known as the *learning curve theory* [119]. Two effects that are reflecting these forces on memory retention are the *forgetting* effect and the *learning* effect.

Modeling the forgetting effect is one of the major challenges that the KT literature has aimed at tackling. Traditional KT models have attempted to incorporate forgetting behavior by adding features such as the number of past trials or the lag time from the previous interaction [78, 135, 139, 154]. In recent years, several deep learning KT models have been developed to take a student's forgetting behavior into consideration during tracing knowledge states.

- **Deep Knowledge Tracing (DKT) + forgetting**

  Nagatani et al. [121] proposed to extend the Deep Knowledge Tracing (DKT) model [137] by adding sequence-related forgetting features. These features include: (1) the number of times a student answers questions with the same KC till the current point of time; (2) the time lapse since the last interaction on a question with the same KC; and, (3) the time lapse since the last interaction on a question regardless of its relating KC. The first feature reflects the learning effect while the other two features reflect the forgetting effect. Different from the previous traditional KT models that use forgetting features only with regard to questions with the same KC [78, 135, 139, 154], this work considers a student's interactions in the whole sequence so as to model more complex forgetting behavior.

- **Knowledge Proficiency Tracing (KPT)**

  Knowledge Proficiency Tracing (KPT) was proposed by Chen et al. [24], which is a probabilistic matrix factorization model that leverages prior information for knowledge tracing. Specifically, two kinds of priors have been considered in this model: (1) *question priors*: the model uses a Q-matrix which was marked by experts to depict the relationship between questions and KCs for generating question representations; and, (2) *student priors*: the model captures the changes in a student's knowledge state over time by jointly applying both learning curve and forgetting curve theories. The learning and forgetting factors are designed based on the assumption that a student's current knowledge state is mainly influenced by two underlying reasons: (a) the more exercises she does, the higher level of related knowledge state she will get; and, (b) the more the time passes, the more knowledge she will forget.

  To further improve the predictive performance, an improved version of KPT, called *Exercise-correlated Knowledge Proficiency Tracing* (EKPT) [67] was later developed, which incorporates the connectivity among questions over knowledge concepts into the probabilistic modeling.

- **HawkesKT**

  Inspired by the Hawkes process [61], Wang et al. [180] proposed *HawkesKT*, a model that uses the point process to adaptively model temporal cross-effects in KT. It assumes that the mastery of a KC by a student is not only affected by previous interactions on

questions of the same KC but also interactions on the other questions (*cross-effects*). Further, the model assumes that cross effects caused by different previous interactions may also have different temporal evolutions on the mastery of different KCs. Although cross effects all decay with time, their decay rates differ from each other because some KCs may be easier to forget than others.

- **Deep Graph Memory Network (DGMN)**

  Deep Graph Memory Network (DGMN) [2] is a hybrid KT model that combines graph neural networks with memory for forgetting aware knowledge tracing. The model aims at modeling the forgetting behavior over a KC space, which has the advantage to capture indirect relationships between questions. DGMN builds a dynamic graph from a knowledge state memory to capture relationships across KCs. Given a sequence of interactions, DGMN uses an attention mechanism to associate questions to their relevant KCs. Then, it calculates forgetting features over the sequence and fuses question embedding, KC graph embedding, and forgetting features using a gating mechanism. The gating output is used to predict the probability of answering the next question correctly.

- **Learning Process-consistent Knowledge Tracing (LPKT)**

  The LPKT model [155] considers the learning gain of a student during the answer prediction. The learning gain is defined as the change in a knowledge state across a time interval since the last answered the question and integrate the time interval lapse between answering questions into the embedding representation of the exercise sequence. The LPKT model consists of three sequential memory cells: 1) a learning progress gate that projects an embedding for the latest exercise in the sequence considering its tag, time to answer, and time-lapse before answering it, 2) a forgetting gate that gets the latest two learning progress embeddings and projects an output representing the forgetting effect, and 3) a prediction cell that considers the forgetting output and the latest question tag embedding to predict the correct answer.

3.2.3.4 Discussion

Deep learning KT models have demonstrated their great potential in solving the KT problem. Below, we discuss several key aspects that are crucial for being considered in their designs.

- **Knowledge state:** One fundamental assumption underlying each deep learning KT model is whether a knowledge state is considered over a single KC or multiple KCs. Accordingly, modeling a student's knowledge states based on the mastery level of KCs by the student is an important task in designing the deep learning KT models. Generally, from early works such as DKT which uses a hidden state to model knowledge states over a single KC, to later works by memory-augmented KT models (e.g., DKVMN and SKVMN) and by text-aware KT models (e.g., EKT) which use matrices to model knowledge states over multiple KCs, a trend in deep learning KT models is to develop a mechanism that is expressive for dynamically capturing knowledge state representations over complex KCs.

- **KC dependencies:** In a KT task, each question is assumed to associate with a single KC or multiple KCs, which is often provided as prior knowledge such as Q-matrix. One main challenge faced by deep learning KT models is to discover the dependencies among different KCs, for example, one KC requires several other KCs as the prerequisite skills. To address this challenge, two lines of research have been explored in the KT literature, including (1) using an attention mechanism to learn how questions are related to each other in terms of their required KCs; and, (2) using a graph-based learning model such as graph neural networks to learn the relationships between KCs or between questions according to their required KCs.

- **Feature augmentation:** To improve the model performance on KT tasks, additional features such as temporal features relating to forgetting behavior and textual features relating to question texts have been leveraged by a number of deep learning KT models in recent years. On one hand, augmenting additional features can usually lead to a more accurate prediction of student learning performance; on the other hand, the augmentation of such additional features depends on their availability in databases, thus limiting their applicability within specific KT applications.

Table 3.5 summarizes the main characteristics of different KT models across these two broad categories.

3.2.4 Usage Assumptions for KT Methods

We reviewed different categories for KT methods, yet it is essential to understand the assumptions underlying each of them when choosing KT methods for applications. Thus, in the following, we discuss the impact of design assumptions for each KT category on applications such as training data and computational resources.

Starting with traditional KT methods such as BKT and its variants [179], we note that these traditional KT methods have a simplified view of the KT problem including assuming the knowledge state to be a binary random variable or having a single KC for each question, which is mainly to keep the posterior computation tractable. IRT and factor analysis methods [40] assume prior learning ability information for each student to be available in addition to an exercise answering history, and this limits their usage in scenarios where such information is missing such as cold-start ones. Accordingly, these methods are more suitable for simple application scenarios that deal with primitive skills (e.g., early learning problems) and do not include multiple KCs in the learning process. Nonetheless, these methods require less computational overhead in comparison to advanced categories (e.g., deep KT methods) and are self-explainable through the learned state space transition probabilities.

The early work of deep KT methods [137] primarily uses conventional neural sequence models such as RNN or LSTM cells, which limits their ability to model multiple KC relationships and long-term dependencies in an exercise answering sequence. Thus, these deep KT methods are suitable for application scenarios where the number of involved KCs is limited and exercise answering sequences are relatively short. Memory-augmented deep KT methods overcome the limitations of these deep KT categories by facilitating to model multiple KC relationships and longer sequence dependencies via a memory mechanism, yet this comes with

KT Model	Learning Model	Knowledge Component		Knowledge State	Forgetting
		Single	Multiple		
BKT [31]	HMM/BN	✓	-	Binary scalar	×
DBN [76]	DBN	-	✓	Binary vector	×
IRT [54]	LR	✓	-	Real-valued vector	×
HIRT [190]	LR	✓	-	Real-valued vector	×
TIRT [190]	LR	✓	-	Real-valued vector	×
LFA [20]	LR	✓	-	Real-valued vector	×
AFM [21]	LR	✓	-	Real-valued vector	×
PFA [134]	LR	✓	-	Real-valued vector	×
KTM [178]	FM	-	✓	Real-valued vector	×
KTM-DLF [44]	FM	-	✓	Real-valued vector	✓
DKT [137]	RNN/LSTM	✓	-	Hidden state (vector)	×
DKT+ [197]	RNN/LSTM	✓	-	Hidden state (vector)	✓
DKT-DSC [115]	RNN/LSTM	✓	-	Hidden state (vector)	×
DKVMN [201]	KVMN	-	✓	Key-value memory (matrix)	×
SKVMN [1]	LSTM+KVMN	-	✓	Key-value memory (matrix)	×
SAKT [126]	FFN+MSA	-	✓	Attentive embedding (matrix)	×
AKT [45]	FFN+MSA	-	✓	Attentive embedding (vector)	✓
SAINT [26]	FFN+ED+MSA	-	✓	Attentive embedding (matrix)	×
SAINT+ [157]	FFN+ED+MSA	-	✓	Attentive embedding (matrix)	✓
RKT [127]	FFN+MSA	-	✓	Attentive embedding (vector)	✓
CKT [156]	1-D(CNN)	-	✓	Attentive embedding (vector)	×
CoKT [106]	FFN+MSA	-	✓	Attentive embedding (matrix)	×
GKT [122]	GNN	-	✓	Vector	×
GIKT [195]	GNN/RNN	-	✓	Vector	×
SKT [173]	GNN	-	✓	Vector	×
PEBG [105]	RNN/LSTM			Hidden state (vector)	×
EERNN [161]	RNN/LSTM	✓	-	Hidden state (vector)	×
EKT [101]	AM/LSTM	-	✓	Attentive embedding (matrix)	×
AdaptKT [25]	ED/LSTM	-	✓	Hidden state (vector)	×
DKT+forget [121]	RNN/LSTM	✓	-	Hidden state (vector)	✓
KPT [67, 24]	FM	-	✓	Real-valued vector	✓
HawkesKT [180]	FM	-	✓	Real-valued vector	✓
DGMN [2]	GCN/KVMN	-	✓	Key-value memory (matrix)	✓
LPKT [155]	RNN/LSTM	-	✓	Hidden state (vector)	✓

Table 3.5: A summary of descriptive characteristics of main knowledge tracing models, where HMM is an abbreviation for Hidden Markov Mode [33], BN for Bayesian Network [179], DBN for Dynamic Bayesian Network [76], LR for Logistic Regression [191], FM for Factorization Machine [169, 166], GNN for Graph Neural Network [149], FFN for Feed Forward Network [177], KVMN for Key-Value Memory Network [114], ED for Encoder and Decoder model [177], MSA for Multi-head Self-Attention mechanism or variants [177], 1-D(CNN) for 1-D Convolutional Neural Network [81] and AM for Attention Mechanism [7, 177].

an additional memory cost to store memory structures. As a result, this category is suitable where memory and computational resources are available. Attention KT methods provide another alternative for modeling long-term dependencies in an exercise answering sequence with a significant computational overhead to execute self-attention computation in quadratic time. This overhead is further magnified when using multiple attention heads. The added performance value of this category is justifiable given the need to deal with long-term dependencies with multiple KC relationships and the availability of extended computing resources, such as multiple GPU units that could distribute attention head computations in parallel. Graph-based KT methods have the capacity to capture complicated question and KC relationships, yet, the majority of these methods assume that graph data exists in prior, which cannot be guaranteed in many real-world application scenarios. There are automated techniques [122] being studied to learn graph structure directly from exercise answering sequences with an additional computational cost. Finally, text-aware deep KT methods assume the existence of text data for questions and KC tags in order to learn effective embeddings. Nevertheless, some KT datasets do not provide text tags for questions and/or KCs, which will limit the usage of these methods in such application scenarios.

3.3 Knowledge Tracing Datasets

This section presents an overview of the benchmark datasets used in the literature to support the evaluation of KT models. All publicly available datasets were downloaded, inspected, and summarized. Table 3.6 lists the datasets and provides general information such as student interactions, the number of questions, and data availability. More details about the datasets are presented below.

3.3.1 ASSISTments Datasets

The ASSISTments datasets [42, 130] contain longitudinal data collected from the free online tutoring ASSISTment platform[1]. Table 3.7 shows that the ASSISTments datasets are the most popular datasets used to benchmark KT models and the ones containing the most questions in total.

These datasets are composed of grade school math exercises sampled from the *Massachusetts Comprehensive Assessment System* (MCAS)[2] containing different types of questions, such as multiple choice, text, and open-ended questions. There are different versions of the ASSISTments datasets with data collected in different periods. Details about each version are presented below.

- **ASSISTments2009:** This dataset was collected during the school year 2009 – 2010 and, when first released, contained a total of 525, 535 interactions (i.e., student responses to questions in the dataset), including duplicates, as discussed in [193]. The latest updated version of this dataset contains 346, 860 interactions given by 4, 217 students to a total

[1] https://www.assistments.org/
[2] https://www.doe.mass.edu/mcas/testitems.html

Dataset	#Questions	#Students	#Interactions	#KCs	Public available
ASSISTments2009	26,688	4,217	346,860	123	Yes
ASSISTments2012	179,999	46,674	6,123,270	265	Yes
ASSISTments2015	100	19,917	708,631	-	Yes
ASSISTChall	3,162	1,709	942,816	102	Yes
STATICS2011	1,224	335	361,092	80	No
Junyi Academy	722	247,606	25,925,992	41	Yes
Simulated-5 (Synthetic)	50	4,000	200,000	5	Yes
Algebra 2005-2006	1,084	575	813,661	112	Yes
Algebra 2006-2007	90,831	1,840	2,289,726	523	Yes
Bridge to Algebra	19,258	1,146	3,686,871	493	Yes
EdNet-KT1	13,169	784,309	95,293,926	188	Yes
EdNet-KT2	13,169	297,444	56,360,602	188	Yes
EdNet-KT3	13,169	297,915	89,270,654	293	Yes
EdNet-KT4	13,169	297,915	131,441,538	293	Yes
Eedi	27,613	118,971	15,867,850	388	Yes

Table 3.6: Dataset Statistics.

of 26,688 distinct questions and 123 KCs. We note that only two-thirds of the questions (17,751) are annotated with KCs. Questions without assigned KCs are annotated with 'NA' (not available) or have no assigned value ('null') whereas all other questions are annotated with up to four KCs.

Despite the popularity of this dataset, its original version is not reliable as discussed in [201] and the updated 'skill-builder'[1] version is preferred as it fixes data modeling issues and removes duplicated records. It is also noteworthy to mention that results obtained with the different versions of this dataset (or with duplicated records) are often reported in the literature but they should not be directly compared to other approaches [190].

- **ASSISTments2012[2]:** This is the largest version of the ASSISTments datasets consisting of data collected for one year (from Sept 2012 to Oct 2013). Despite the ASSISTments team reporting that the dataset contains approximately 10 million 'exercises', the available dataset consists of 179,999 distinct questions answered by 46,674 students resulting in 6,123,270 interactions. We note that the vast majority (126,908) of the questions does not have any of the 265 KCs associated with them. The lack of questions annotated with KCs may explain the overall lower performance of the KT models when applied to

[1] https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data
[2] https://sites.google.com/site/assistmentsdata/home/2012-13-school-data-with-affect

this dataset (see Table 3.6).

- **ASSISTments2015**[1]**:** This dataset contains 708,631 student interactions with the ASSISTments platform in the year 2015 produced by 19,917 students answering 100 distinct questions. The dataset contains only four attributes: (i) the questions' identifiers; (ii) the identifier of the student who answered the questions; (iii) an attribute indicating the correctness of the answer given by each student; and, (iv) a log attribute where a temporal sequence of the answers given by the students can be inferred.

  Unlike previous versions of the ASSISTments datasets, no metadata and no KC are provided. Another difference between this and previous datasets is related to the average number of responses given to each question. This dataset has an average of 7,086.31 answers per question whereas the 2009 and 2012 versions have 12.99 and 34.01, respectively.

- **ASSISTment Challenge (ASSISTChall)**[2]**:** Released in 2017, the full ASSISTment Challenge dataset contains data from 2004 – 2005 and 2005 – 2006 academic years. It contains 3,162 distinct questions answered by 1,709 students over 102 KCs resulting in 942,816 interactions. On average, this dataset has 298.17 answers per question, placing it second in terms of the answer per question ratio (only lower than the ASSISTments2015 dataset). As part of a data mining competition, this dataset contains the most descriptive information among the ASSISTments datasets.

3.3.2 STATICS2011 Dataset

This dataset is available upon request and contains students' interactions with questions related to the Engineering Statics course[3] taught at the Carnegie Mellon University during Fall 2011 [83].

The original dataset contains 361,092 interactions, 335 students, and 1,224 questions. In the KT literature, this dataset is often preprocessed [201] resulting in 1,223 distinct questions answered by 333 students over 80 KCs. After preprocessing, the number of students' interactions is almost halved to 189,297. This preprocessing was justified due to a large number of interactions without information on whether the questions have been correctly answered. The preprocessing considers the concatenation of the attributes 'problem name' and 'step name' and only the interactions with a valid first attempt. On average, this dataset has 568.45 answers per question.

[1] https://sites.google.com/site/assistmentsdata/home/2015-assistments-skill-builder-data
[2] https://sites.google.com/view/assistmentsdatamining
[3] https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=507

3.3.3 Junyi Academy Dataset

This dataset[1] was collected between November 2010 and March 2015 from the Junyi Academy [23], an e-learning platform in Taiwan. The original dataset contains 25,925,992 interactions, 247,606 students, 722 distinct questions, and 41 KCs covering a number of topics in math. However, considering the same preprocessing step made in the previous datasets (i.e., students' interactions with no hints given to help solve the questions and only students who have attempted each question once), the number of interactions drops to 21,571,469 ($\sim$ 17%), 220,441 ($\sim$ 11%) students, and 716 ($<$ 1%) distinct questions. Finally, on average, this dataset has 97.85 answers per question.

Although this dataset has been commonly used in the KT literature, the performance reported in some of the works cannot be directly compared. This is because these works use different subsets or preprocessing techniques [1, 127, 173]. Further, note that an updated version of the Junyi dataset is available in Kaggle[2] with data collected from August 2018 to August 2019. This dataset contains 11,468,379 interactions, 25,649 students, and 1,701 distinct questions where no hints were given and students have attempted each question only once.

3.3.4 Simulated-5 (Synthetic) Dataset

This synthetic dataset was proposed by Piech et al. [137], which simulates virtual students answering the same sequence of questions over a set of KCs in a controlled environment[3]. The dataset is divided into two subsets, including training and testing. Each subset contains 50 distinct questions associated with a single KC and a difficulty level. In total, each question is answered by 4,000 virtual students resulting in 200,000 interactions. The classic Item Response Theory [43] was used to create the interactions and simulate students learning over time [137, 197]. This dataset provides a standard format and does not require preprocessing steps such as removing duplicates or steps to infer a sequence of questions answered by a student, potentially allowing direct comparison between different KT models.

3.3.5 KDDcup Dataset

This dataset was presented at the KDDcup 2010 Educational Data Mining challenge[4] [160] and contains 13–14 year old students' responses to Algebra questions from 2005 to 2007 extracted from the intelligent tutoring system called "The Cognitive Tutors" developed by Carnegie Learning Inc. in the US. The dataset is split into three subsets described in what follows.

- **Algebra 2005-2006:** Considering both training and test data, collected between August 2005 and June 2006, this dataset contains 1,084 distinct questions answered by 575 students resulting in 813,661 interactions. Unlike other datasets, each question is divided

[1] https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=1198
[2] https://www.kaggle.com/junyiacademy/learning-activity-public-dataset-by-junyi-academy/tasks
[3] https://github.com/chrispiech/DeepKnowledgeTracing/tree/master/data/synthetic
[4] https://pslcdatashop.web.cmu.edu/KDDCup

into sub-questions totaling 210,136 sub-items. On average, there are 750.60 answers per question. The total number of distinct KCs is 112 and each sub-question is associated with one or more KCs. As in previous datasets, the total number of interactions drops to 57.8%, the number of students roughly remains the same (574) and the number of sub-items is reduced to 130,256 if the data is preprocessed removing sub-items with no KCs assigned, interactions where students used any hints and questions with two or more attempts.

- **Algebra 2006-2007:** This subset contains 2,289,726 interactions generated by 1,840 students answering 90,831 distinct questions with a total of 577,119 sub-items classified by one or more of the 523 KCs. On average, there are 49.36 answers per question. Similarly to the Algebra 2005-2006 dataset, the number of interactions drops to 1,567,072, the number of students falls to 1,813 and the number of questions and sub-items reduce to 89,877 and 470,187, respectively, after the data preprocessing step. After inspecting this subset, we found that the timestamps provided are incorrect. This may affect the prediction of KT approaches and may explain the low adoption of this subset in comparison to the others.

- **Bridge to Algebra:** This subset contains 3,686,871 interactions collected between November 2006 and June 2007 generated by 1,146 students, 19,258 distinct questions divided into 207,790 sub-items classified by one or more of the 493 KCs. After preprocessing, the total number of interactions is almost halved (1,721,987), the number of questions drops (18,715) slightly, whereas the number of items is reduced to 123,778 ($\sim 40\%$) and the number of students remains almost the same (1,145).

3.3.6 EdNet Dataset

EdNet[1] is a hierarchical dataset composed of four subsets identified by the ids: 'KT1', 'KT2', 'KT3' and 'KT4', each containing different types of student activities. 'KT1', for example, contains question-response pairs similar to other datasets. The main difference is that some questions in this dataset are organized in bundles (a set of questions that must be completed altogether). This dataset contains over 95,293,926 interactions, 13,169 questions, 784,309 students, and 188 KCs.

Unlike the other datasets, 'KT2' contains the actions of the users during question-solving activities. For example, it records the final submission and student decision-making (alternating choices) before submitting the final answer. This subset has 56,360,602 interactions and 297,444 students. Besides the actions in 'KT1' and 'KT2', 'KT3' subset includes information about how students interact with learning activities to answer a question (e.g., watch a lecture). This subset contains more KCs (293), interactions (89,270,654), and students (297,915). Finally, 'KT4' is the most complete subset containing every action recorded by the EdNet system, including students' purchases (e.g., course purchases). This subset contains 131,441,538 interactions in total.

[1] https://github.com/riiid/ednet

Overall, the EdNet dataset series incrementally provides information about student activities and behaviors. The dataset was collected over two years from the intelligent online tutoring platform named *Riid TUTOR*[1] dedicated to practicing English for international communication (TOEIC) assessment [27] in South Korea. The variety of recorded behaviors and the large size of data points are unique aspects of this dataset.

3.3.7 Eedi Dataset

The Eedi dataset [2] [184] was released for the NeurIPS Education Challenge [185] in 2021. It comprises a comprehensive collection of mathematics question logs from 118,971 students. The students span across various educational levels, ranging from primary schools to high schools. The dataset was collected over a period of two years, providing substantial temporal coverage. This dataset contains over 15,867,850 interactions, 27,613 questions, 118,971 students, and 388 KCs. Overall, the Eedi dataset presents a diverse collection of mathematics question logs, along with accompanying demographic details.

3.3.8 Limitations of Existing Datasets

We note that common limitations across the reviewed KT datasets can be summarized as missing the following 1) question tag text in the recorded data, 2) KC-KC relationships, 3) ground truth difficulty of questions, 4) recording answer-related facts such as time taken to answer or the number of choice selection changes, and 5) student's feedback during practice such as their perceived question difficulty or their confidence in their answer. The proposed DBE-KT22 addresses all of these limitations to facilitate the usage of advanced KT techniques and enables a variety of machine learning tasks to be evaluated, such as natural language processing on question and KC textual data, graph modeling on question and KC relational data, curriculum learning based on question difficulty data, or learning cognitive analysis on student's recorded feedback during practice.

3.3.9 Discussion

Table 3.7 presents the results obtained by several KT models using the aforementioned datasets. The results are reported using the AUC-ROC curve — a traditional performance measurement for binary classification models. The probabilistic receiver operating characteristic (ROC) curve plots the true positive rate (TPR) against the false positive rate (FPR) while the area under the curve (AUC) reports how well a KT model can distinguish between correct and incorrect answers. The AUC-ROC ranges from 0 to 1, where 0.5 indicates an uninformative classifier (random guesses) and 1 is a perfect classifier.

It is important to note that the results cannot be directly compared if experimental settings are not standardized. As discussed in [190], the results obtained without removing duplicate interactions (preprocessing steps) may be inaccurate and appear elevated. The existence of duplicates is also acknowledged in the KDDcup dataset[3]. We reported in the previous sections

[1] https://company.riiid.co/en/product

[2] https://eedi.com/projects/neurips-education-challenge

[3] https://pslcdatashop.web.cmu.edu/KDDCup/FAQ/

the raw number of interactions, students, KCs, etc., and whenever different, the total after removing duplicates and discarding noise data.

The lack of accurate and descriptive information about each of the attributes in the datasets also hinders experiments. An example is in the ASSISTments datasets where terminology used is confusing[1] or lacking[2]. This may explain the different AUC values reported to the same approach and dataset pairs presented in Table 3.7.

The sequence of students' interactions is also an important component of a KT task that is affected by the quality of the datasets. Due to noise in the data (e.g., incorrect timestamps, null values, etc.), the sequence of interactions may not be correctly extracted from the datasets which may accordingly impact the performance of the proposed KT models. For example, after inspecting the datasets, we identified timestamp errors in the Algebra 2006 – 2007, which may justify its low adoption in comparison to the other two Algebra datasets.

The fact that the benchmark datasets are often used for multiple tasks other than the KT problem hinders the correct use and interpretation of the datasets in the KT context. The works in [115, 197, 195, 201, 126, 45, 180], for example, report different numbers of knowledge components for the same datasets and the work in [45] discusses how different experimental settings (association between KC and questions) may impact the reported results. The data model and the file format was chosen to represent the datasets may also contribute to misinterpretation of the data. For example, the data is often extracted from relational databases and stored in comma-separated values (CSV), compressing information in a single attribute using non-standard characters, e.g., the KDDcup dataset uses double tilde characters to assign multiple KCs to a question. Given the hierarchical and relational nature of the data, XML and JSON are more suitable and explicit formats.

Another critical aspect is the lack of more diverse datasets. The existing public datasets are often from a specific domain (mainly math) and a specific region (e.g., the ASSISTment datasets from the US, and EdNet, the most recent publicly available dataset, from South Korea). Most of the datasets do not provide demographic information, and therefore gender-based, or other similar predictions cannot be performed.

Finally, the benchmark datasets have been updated over the years, and the version used by the KT models is not readily identifiable, which can compromise their direct comparison. A version control mechanism for the publicly available datasets would help keep track of every change made to a dataset over time and allows for consistent and comparable results.

[1] https://sites.google.com/site/assistmentsdata/home/2012-13-school-data-with-affect
[2] https://sites.google.com/site/assistmentsdata/home/2015-assistments-skill-builder-data

Model	Data Split Ratio (Train-Test)	Dataset									
		ASSISTments				STATICS	Junyi	Synthetic	KDDcup		EdNet
		2009	2012	2015	Chall				Algebra 2005-2006	Bridge 2006-2007	
BKT [31]	80%-20%	0.651 [115] 0.67 [137] 0.6571 [195]	0.623 [115] 0.6204 [195]	0.611 [115] 0.678 [173]	-	-	0.68 [137] 0.831 [173]	0.54 [137]	0.642 [115]	-	0.6027 [195]
IRT [54]	80%-20%	0.7651 [190] 0.6908 [178] 0.751 [115] 0.5869 [180]	0.702 [44] 0.743 [115] 0.6340 [180]	0.672 [115]	-	-	-	-	0.771 [44] 0.806 [115]	0.747 [44] 0.8542 [190]	-
HIRT [190]	80%-20%	0.774 [190]	-	-	-	-	-	-	-	0.8597 [190]	-
TIRT [190]	80%-20%	0.7653 [190]	-	-	-	-	-	-	-	0.8542 [190]	-
AFM [21]	80%-20%	0.6163 [178]	0.610 [44]	-	-	-	-	-	0.707 [44]	0.706 [44]	-
PFA [134]	80%-20%	0.6849 [178] 0.703 [115]	0.670 [115] 0.669 [44]	0.689 [115]	-	-	-	-	0.760 [115] 0.744 [44]	0.746 [44]	-
KTM [178]	80%-20%	0.8186 [178] 0.7169 [195] 0.7425 [180]	0.7535 [180] 0.6788 [195]	-	-	-	-	-	-	-	0.6888 [195]
KTM-DLF [44]	80%-20%		0.756 [44]	-	-	-	-	-	0.837 [44]	0.812 [44]	-
DKT [137]	80%-20%	0.7429 [190] 0.86 [137] 0.8212 [197] 0.721 [115] 0.820 [126] 0.817 [45] 0.709 [122] 0.7561 [195] 0.7515 [180]	0.713 [115] 0.712 [127] 0.7286 [195] 0.7235 [121] 0.7308 [180]	0.707 [115] 0.731 [45] 0.736 [126] 0.7365 [197] 0.727 [173]	0.7263 [45] 0.734 [126] 0.7343 [197]	0.8233 [45] 0.815 [126] 0.8159 [197]	0.814 [127] 0.85 [137] 0.847 [173]	0.8255 [197] 0.823 [126] 0.75 [137]	0.784 [115]	0.751 [122] 0.8901 [190]	0.6822 [195] 0.7638 [157]
DKT+ [197]	80%-20%	0.8227 [197] 0.8024 [45] 0.822 [126]	-	0.7371 [197] 0.728 [173] 0.737 [126] 0.7313 [45]	0.7343 [197] 0.728 [126] 0.7124 [45]	0.8349 [197] 0.835 [126] 0.8301 [45]	0.889 [173]	0.8264 [197] 0.824 [126]	-	-	-
DKT-DSC [115]	80%-20%	0.735 [115]	0.721 [115]	0.716 [115]	0.430 [115]	-	-	-	0.792 [115]	-	-
DKVMN [201]	70%-30%	0.8157 [1] [201]	-	0.7268 [1] [201]	-	0.8284 [1] [201]	0.8027 [1]	0.8273 [1] [201]	-	-	-
SKVMN [1]	70%-30%	0.8363 [1]	-	0.7484 [1]	-	0.8485 [1]	0.8267 [1]	0.840 [1]	-	-	-
SAKT [126]	80%-20%	0.848 [126] 0.752 [45] 0.6860 [180]	0.735 [127] 0.6906 [180]	0.854 [126] 0.7212 [45]	0.734 [126] 0.6569 [45]	0.853 [126] 0.8029 [45]	0.834 [127]	0.832 [126]	-	-	0.7671 [26] 0.7663 [157]
AKT [45]	80%-20%	0.8346 [45] 0.7474 [180]	0.7555 [180]	0.7828 [45]	0.7702 [45]	-	-	-	-	-	-
SAINT [26]	80%-20%	-	-	-	-	-	-	-	-	-	0.7811 [26] 0.7816 [157]
SAINT+ [157]	80%-20%	-	-	-	-	-	-	-	-	-	0.7914 [157]
RKT [127]	80%-20%	-	0.793 [127]	-	-	-	0.860 [127]	-	-	-	-
GKT [122]	80%-20%	0.723 [122]	0.735 [173]	-	-	-	0.893 [173]	-	-	0.769 [122]	-
GIKT [195]	80%-20%	0.7896 [195]	0.7754 [195]	-	-	-	-	-	-	-	0.7523 [195]
SKT [173]	80%-20%	-	-	0.746 [173]	-	-	0.908 [173]	-	-	-	-
DKT+forget [121]	80%-20%	0.754 [180]	0.722 [127] 0.7309 [121] 0.7462 [180]	-	-	-	0.840 [127]	-	-	-	-
HawkesKT [180]	80%-20%	0.763	0.768	-	-	-	-	-	-	-	-
DGMN [2]	70%-30%	86.1 [2]	-	-	-	86.4 [2]	-	85.9 [2]	83.4 [2]	-	-

Table 3.7: AUC results reported by different KT models.

3.4 Summary

In this chapter, we presented a comprehensive review of the knowledge tracing literature and datasets. We proposed a categorization that divides knowledge tracing methods into two broad categories: traditional methods covering Bayesian and factor model approaches, and deep learning methods covering recent state-of-the-art techniques, including recurrent, memory, attention, and graph approaches. For each category, we highlighted the main characteristics, including assumptions, governing factors for applications, and their strengths and weaknesses. We identified he key datasets used in the knowledge tracing literature along with the main characteristics of each dataset. We summarized all the reported performance results for each dataset in one table to provide a holistic view of the current state. Moreover, we discussed the application assumptions for different KT categories.

Chapter 4

Knowledge Tracing with Sequential Key-Value Memory Networks

4.1 Overview

Inspired by recent advances in deep learning [92], several deep learning KT models have been proposed. A pioneer work by Piech et al. [137] reported *Deep Knowledge Tracing* (DKT), which uses Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) units [51, 163] to predict student performance on new exercises given their past learning history. In DKT, a student's knowledge states are represented by a sequence of hidden states that successively encode relevant information from past observations over time. Although DKT has achieved substantial improvements in prediction performance over BKT[31], due to the limitation of representing a knowledge state by one hidden state, it lacks the ability to go deeper to trace how specific concepts are mastered by a student (i.e., concept states) in a knowledge state. To deal with this limitation, *Dynamic Key-Value Memory Networks* (DKVMN) [201] was proposed to model a student's knowledge state as a complex function over all underlying concept states using a key-value memory. Their idea of augmenting DKVMN with an auxiliary memory follows the concepts of Memory-Augmented Neural Networks (MANN) [148, 53]. However, DKVMN acquires knowledge growth through the most recent exercise and thus fails to capture long-term dependencies in an exercise sequence (i.e., relevant past experience to a new observation).

In this chapter, we present a new KT model called *Sequential Key-Value Memory Networks* (SKVMN) that provides three advantages over the existing deep learning KT models as follows:

- First, SKVMN unifies the strengths of both the recurrent modeling capacity of DKT and the memory capacity of DKVMN for modeling student learning. We observe that although a key-value memory can help trace the concept states of a student, it is not effective in modeling long-term dependencies on sequential data. We remedy this issue by incorporating LSTMs into the sequence modeling for a student's knowledge states over time. Thus, SKVMN is not only augmented with a key-value memory to enhance the representation capability of knowledge states at each time step but also can provide recurrent modeling capability for capturing dependencies among knowledge states at

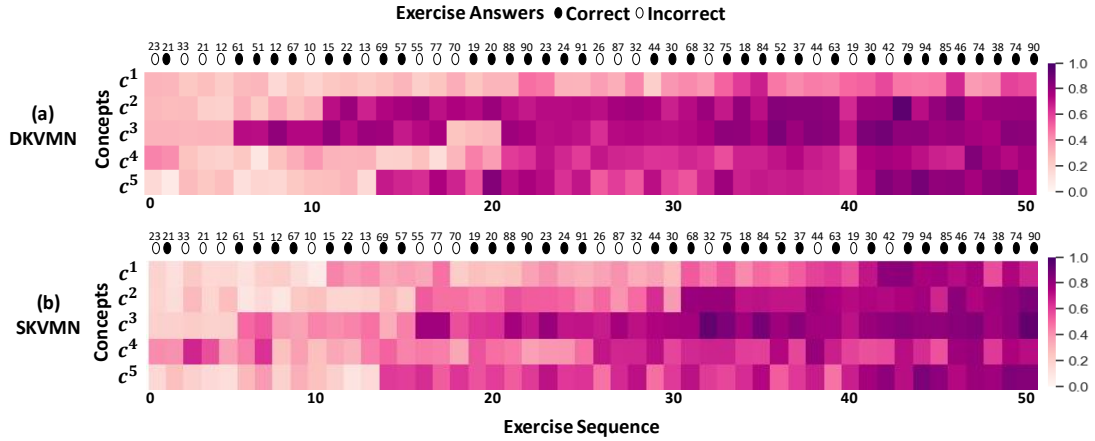


Figure 4.1: An illustration of how a student's knowledge states are evolving for a sequence of 50 exercises in the dataset ASSISTments2009 using: (a) DKVMN and (b) SKVMN. There are 5 concepts $\{c^1, \ldots, c^5\}$ underlying these 50 exercises. Each row depicts the evolution of the concept states for a specific concept, and each column depicts the knowledge state at a certain time step.

different time steps in a sequence.

- Second, SKVMN uses a modified LSTM with hops, called *Hop-LSTM*, in its sequence modeling. Hop-LSTM deviates from the standard LSTM architecture by using a triangular layer for discovering sequential dependencies between exercises in a sequence. Then, the model may hop across LSTM cells according to the relevancy of the latent learning concepts. This enables relevant exercises that correlate to similar concepts to be processed together. In doing so, the inference becomes faster, and the capacity of capturing long-term dependencies in an exercise sequence is enhanced.

- Third, SKVMN improves the write process of DKVMN in order to represent better knowledge states stored in a key-value memory. In DKVMN, the current knowledge state is not considered when calculating the knowledge growth of a new exercise. This means that the previous learning experience is ignored. For example, when a student attempts the same question multiple times, the same knowledge growth would be added to the knowledge state, regardless of whether the student has previously answered this question or how many times the answers were correct. SKVMN solves this issue by using a summary vector as input for the write process, which reflects both the current knowledge state of a student and the prior difficulty of a new question.

We have extensively evaluated our proposed model SKVMN [1] on five well-established KT benchmark datasets and compared it with the state-of-the-art KT models. The experimental results show that (1) SKVMN outperforms DKT and DKVMN, on all five datasets, (2) SKVMN can better discover the correlation between latent concepts and questions, and (3) SKVMN captures the dynamics of student's knowledge state over multiple latent concepts, and leverages sequential dependencies between exercises in an exercise sequence for improved prediction accuracy.

[1] Source code https://pykt-toolkit.readthedocs.io/en/latest/_modules/pykt/models/skvmn.html#SKVMN

Figure 4.1 illustrates how a student's knowledge states evolve as the student attempts a sequence of 50 exercises in DKVMN and SKVMN. We can see that compared with the knowledge states of DKVMN depicted in Figure 4.1.(a), our model SKVMN provides a smoother transition between two successive concept states as depicted in Figure 4.1.(b). Moreover, SKVMN captures a smooth, progressive evolution of knowledge states over time (i.e., through this exercise sequence), which more accurately reflects the way a student learns (will be discussed in detail in Section 4.5.4).

The remainder of this chapter is organized as follows. Section 4.2 defines the knowledge tracing problem. Section 4.3 presents our proposed KT model SKVMN. Sections 4.4 and 4.5 discuss the experimental design and results. We conclude this chapter in Section 4.6.

4.2 Problem Definition

In addition to the generic definition of the knowledge tracing problem that we introduced in Section 1.1.2, we address a subproblem of identifying questions in the past exercise answering sequence X that are relevant to the input question at time point t. These relevant questions form a subset of X that can be used for distilling relevant parts of a student's knowledge state to the input question q_t. We define this subproblem as given the past exercise answering sequence X; we want to learn a question relevancy membership function $\mu_\omega : (X, q_t) \rightarrow \hat{X}$ parameterized by ω, where $\hat{X} \subset X$. We learn μ_ω with the KT hypothesis function $f_\theta : (q_t, \hat{X}) \rightarrow \hat{y}_t$ in an end-to-end fashion by minimizing the following answer prediction loss $\mathcal{L}$ function:

$$\underset{\theta^* \in \Theta, \omega^* \in \Omega}{\arg\min} \overbrace{\mathcal{L}(f_\theta(q_t, \underbrace{\mu_\omega(X, q_t)}_{\text{Membership}}))}^{\text{Answer Prediction}}, y_t) \tag{4.1}$$

4.3 Methodology

In this section, we introduce our model *Sequential Key-Value Memory Networks* (SKVMN) and its building blocks.

We first present an overview of SKVMN. Then, we show how a key-value memory can be attended, read, and written in our model. To leverage sequential dependencies among latent concepts for prediction, we present modified LSTMs, called *Hop-LSTMs*. Lastly, we discuss the optimization techniques used in the model.

4.3.1 Model Overview

The SKVMN model is augmented with a key-value memory $\langle \mathbf{M}^k, \mathbf{M}^v \rangle$ following the work in [201], i.e., a pair of one static matrix $\mathbf{M}^k$ of size $N \times d_k$, called the *key matrix*, and one dynamic matrix $\mathbf{M}^v$ of size $N \times d_v$, called the *value matrix*. Both the key matrix and the value matrix have the same N memory slots, but they may differ in their state dimensions d_k and d_v. The key matrix stores the latent concepts underlying questions, and the value matrix stores the

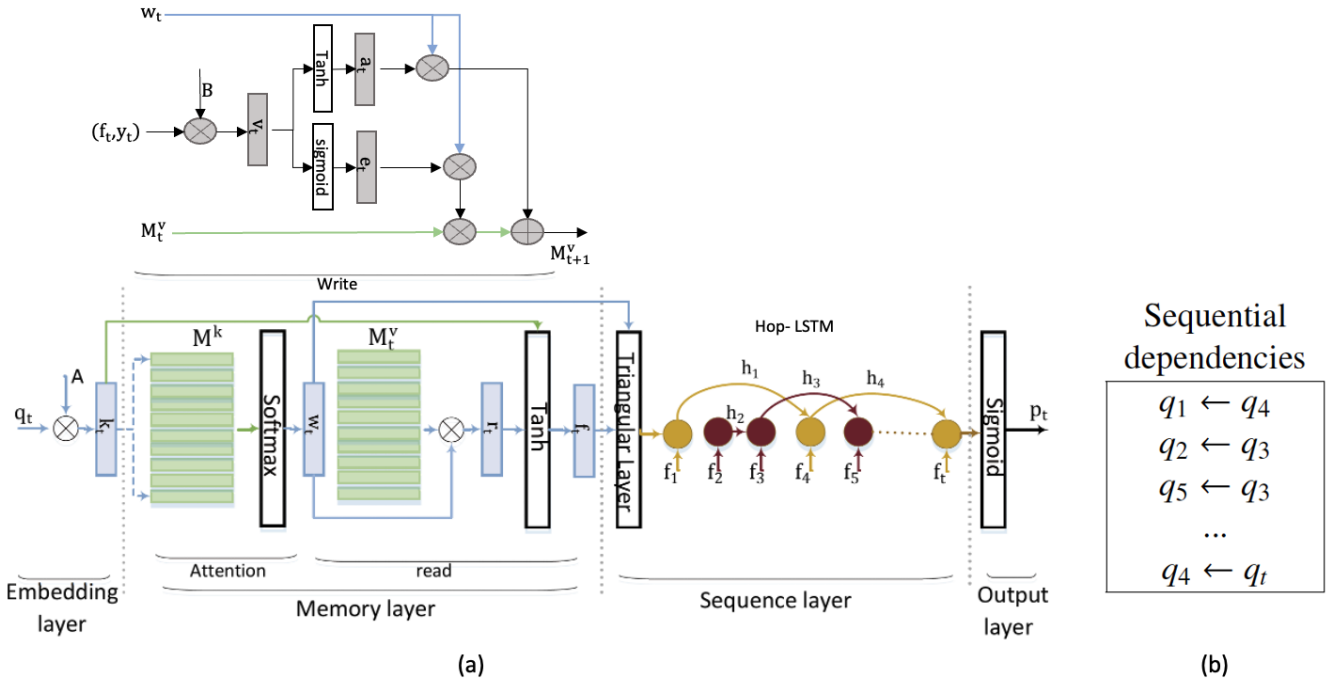


Figure 4.2: An illustration of the SKVMN model at time step t, where q_t is the input question, the exercise history $X = \langle (q_1, y_1), (q_2, y_2), \ldots, (q_{t-1}, y_{t-1}) \rangle$, and each f_i is the summary vector of the question q_i for $i \in [1, t]$: (a) the model has four layers, namely the embedding, memory, sequence, and output layers; and (b) sequential dependencies among questions in X.

concept states of a student (i.e., the knowledge state) which can be changed dynamically based on student learning.

Given an input question q_t at time step t, the SKVMN model retrieves the knowledge state of a student from the key-value memory $\langle \mathbf{M}^k, \mathbf{M}^v \rangle$, and predicts the probability of correctly answering the question q_t by the student. Figure 4.2.(a) illustrates the SKVMN model at time step t, which consists of four layers: the embedding, memory, sequence, and output layers.

- The embedding layer maps an input question at time step t into a high-dimensional vector space. We discuss the details of the layer in Section 4.3.2.
- The memory layer involves two processes: attention and read, where the attention process provides an addressing mechanism for the input question q_t to allocate the relevant information from the key-value memory, and the read process uses the attention vector to retrieve the current knowledge state of the student from the value matrix $\mathbf{M}^v_t$. The details of the attention and read processes will be discussed in Sections 4.3.3.1 and 4.3.3.2. After a student has attempted the input question q_t with the answer y_t, the value matrix in the key-value memory of the SKVMN model needs to be updated to reflect the latest knowledge state of the student. Figure 4.2.(a) depicts how the value matrix is transited from $\mathbf{M}^v_t$ at time step t to $\mathbf{M}^v_{t+1}$ at time step $t+1$ using the write process. The details of the write process are discussed in Section 4.3.3.3.
- The sequence layer consists of a set of recurrently connected LSTM cells, where LSTM cells are connected based on their sequential dependencies determined by a Triangular layer as depicted in Figure 4.2.(b). The details of the sequence layer are discussed in Section 4.3.4.
- The output layer generates the probability of correctly answering the input question q_t. The details of this layer are presented in Section 4.3.5

4.3.2 Embedding Layer

Given a question $q_t \in Q$ as input, the model represents q_t as a "one-hot" vector of length |Q| in which all entries are zero, except for the entry that corresponds to q_t. In order to map q_t into a continuous vector space, q_t is multiplied by an embedding matrix $\mathbf{A} \in \mathbb{R}^{|Q| \times d_k}$, which generates a continuous embedding vector $k_t \in \mathbb{R}^{d_k}$, where d_k is the embedding dimension.

4.3.3 Memory Layer

There are three processes relating to access to a key-value memory in our model: attention, read and write. In the following, we elaborate on these processes.

4.3.3.1 Attention

An attention vector w_t is obtained by applying the Softmax function to the inner product between the embedding vector k_t and each key slot $\mathbf{M}^{k(i)}$ in the key matrix $\mathbf{M}^k$ as follows:

$$w_t(i) = \text{Softmax}(k_t^T \mathbf{M}^k(i)) \tag{4.2}$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$. Conceptually, w_t represents the correlation between the question q_t and the underlying latent concepts stored in the key matrix $\mathbf{M}^k$.

4.3.3.2 Read

For each exercise, $x_t = (q_t, y_t)$, the model uses its corresponding attention vector w_t to retrieve the concept states of the student with regard to the question q_t from the value matrix $\mathbf{M}_t^v$. Specifically, the read process takes the attention vector w_t as input and yields a read vector r_t, which is the weighted sum of all values being attended by w_t in the memory slots of the value matrix $\mathbf{M}_t^v$, i.e.,

$$r_t = \sum_{i=1}^{N} w_t(i) \mathbf{M}_t^{v(i)} \tag{4.3}$$

The read vector r_t is concatenated with the embedding vector k_t. Then, the combined vector is fed to a Tanh layer to calculate the summary vector f_t:

$$f_t = \text{Tanh}(\mathbf{W}_1^T [r_t, k_t] + b_1) \tag{4.4}$$

where $\text{Tanh}(z_i) = (e^{z_i} - e^{-z_i}) / (e^{z_i} + e^{-z_i})$, $\mathbf{W}_1$ is the weight matrix of the Tanh layer, and b_1 is the bias vector. While the read vector r_t represents the student's knowledge state with respect to the relevant concepts of the current question q_t, the summary vector f_t adds prior information of the question (e.g., the level of difficulty) to this knowledge state. Intuitively, f_t represents how well the student has mastered the latent concepts relevant to the question q_t before attempting this question.

4.3.3.3 Write

The write process occurs each time after the student has attempted a question. The purpose of this write process is to update the concept states of the student in the value matrix $\mathbf{M}_t^v$ using the knowledge growth gained through attempting the question q_t. This update leads to the transition of the value matrix from $\mathbf{M}_t^v$ to $\mathbf{M}_{t+1}^v$ as depicted in Figure 4.2.(a).

To calculate the knowledge growth, our model considers not only the correctness of the answer y_t for q_t, but also the student's mastery level of the concept states f_t before attempting q_t. Each (f_t, y_t) is represented as a vector of length $2|Q|$ and multiplied by an embedding matrix $\mathbf{B} \in \mathbb{R}^{2|Q| \times d_v}$ to get a write vector v_t, which represents the knowledge growth of the student obtained by attempting the question.

Similar to other memory-augmented networks [53, 201], the write process proceeds with two gates: *erase gate* and *add gate*. The former controls what information to erase from the knowledge state after attempting the latest exercise x_t, while the latter controls what information to add to the knowledge state. From the knowledge tracing perspective, these two gates capture the forgetting and enhancing aspects of learning knowledge, respectively.

With the write vector v_t for the knowledge growth, an erase vector e_t is calculated as:

$$e_t = \text{sigmoid}(\mathbf{W}_e^T \cdot v_t + b_e) \tag{4.5}$$

where $\text{Sigmoid}(z_i) = 1/(1+e^{-z_i})$ and $\mathbf{W}_e$ is the weight matrix. Using the attention vector w_t, the value matrix $\tilde{\mathbf{M}}^v_{t+1}$ after applying the erase vector e_t is

$$\tilde{\mathbf{M}}^{v(i)}_{t+1} = \mathbf{M}^v_t(i)[1 - w_t(i)e_t]. \tag{4.6}$$

Then, an add vector a_t is calculated by

$$a_t = \text{Tanh}(\mathbf{W}^T_a \cdot v_t + b_a) \tag{4.7}$$

where $\mathbf{W}_a$ is the weight matrix. Finally, the value matrix $\mathbf{M}^v_{t+1}$ for the next knowledge state is updated as:

$$\mathbf{M}^v_{t+1}(i) = \tilde{\mathbf{M}}^v_{t+1}(i) + w_t(i)a_t \tag{4.8}$$

4.3.4 Sequence Layer

Now we discuss the sequence modeling approach used at the sequence layer for predicting the probability of correctly answering the question q_t based on the exercise history.

4.3.4.1 Sequential Dependencies

An exercise history X of a student may contain a long sequence of exercises. For example, the average sequence length in the ASSISTments2009 dataset is 233 ± 100 questions per sequence. However, as different exercises may correlate to different latent concepts, not all exercises in X can equally contribute to the prediction of answering a given question q_t. Thus, we observe that, by hopping across irrelevant exercises in X with regard to q_t, recurrent models can be applied on a shorter and more relevant sequence, leading to more efficient and accurate prediction performance.

For each question in Q, since its attention vector reflects the correlation between this question and the latent concepts in C, we consider that the similarity between two attention vectors can provide a good indication of how their corresponding questions are relevant in terms of their correlations with latent concepts. For example, suppose that we have three latent concepts $C = \{c_1, c_2, c_3\}$, and two attention vectors $w_1 = [0.15, 0.25, 0.6]^T$ and $w_2 = [0.2, 0.3, 0.5]^T$ that correspond to the questions q_1 and q_2, respectively. Then q_1 and q_2 are considered as being relevant if both w_1 and w_2 are mapped to a vector $[0, 0, 1]^T$ where 0, 1 and 2 refer to the value ranges “low", “medium" and “high", respectively.

Now, the question is: how to identify similar attention vectors? For this, we use the triangular membership function [82]:

$$\mu(x) = max(min(\frac{x-a}{b-a}, \frac{c-x}{c-b}), 0), \tag{4.9}$$

where the parameters a and c determine the feet and the parameter b determines the peak of a triangle. We use three triangular membership functions for three value ranges: low (0), medium (1), and high (2). Each real-valued component in an attention vector is mapped to one of the three value ranges. Each attention vector w_t is associated with an identity vector d_t.

Similar attention vectors have the same identity vector, while dissimilar attention vectors have different identity vectors.

Then, at each time step t, for the current question q_t and an exercise history X, we say q_t is *sequentially dependent* on $q_{t-\lambda}$ in X with $\lambda \in (0, t)$, denoted as $q_{t-\lambda} \leftarrow q_t$, if the following two conditions are satisfied:

- The attention vectors of $q_{t-\lambda}$ and q_t have the same identity vector, i.e., $d_{t-\lambda} = d_t$, and
- There is no other q_j in X such that $d_j = d_t$ and $j > t - \lambda$, i.e., $q_{t-\lambda}$ is the most recent exercise in X that is relevant to q_t.

Over time, given a sequence of questions $\langle q_1, q_2, \ldots, q_{|Q|} \rangle$, we thus have an exercise history X′ in which exercises are partitioned into a set of subsequences $\{X'_1, \ldots, X'_n\}$ with $\Sigma_{k=1}^{n} |X'_k| = |X'|$ and, for any two consecutive exercises (q_i, y_i) and (q_j, y_j) in the subsequence X'_k, q_j is sequentially dependent on q_i, i.e. $q_i \leftarrow q_j$. In Figure 4.2, based on the sequential dependencies presented in Figure 4.2.(b), X is partitioned into $\langle q_1, q_4, \ldots, q_t \rangle$ and $\langle q_2, q_3, q_5, \ldots \rangle$, which correspond to the recurrently connected LSTM cells depicted in Figure 4.2.(a). We will discuss further details in the following.

4.3.4.2 Hop-LSTM

Based on sequence dependencies between exercises in a sequence, a modified LSTM with hops, called Hop-LSTM, is used to predict the probability of correctly answering a new question by a student. Unlike the standard LSTM architecture, Hop-LSTM allows us to recurrently connect the LSTM cells for questions based on the relevance of their latent concepts. More precisely, two LSTM cells in Hop-LSTM are connected only if the input question of one LSTM cell is sequentially dependent on the input question of the other LSTM cell. This means that Hop-LSTM has the capability of hopping across the LSTM cells when their input questions are irrelevant to the current question q_t.

Formally, at each time step t, for the question q_t, if there is an exercise $(q_{t-\lambda}, y_{t-\lambda}) \in X$ with $\lambda \in (0, t)$ and $q_{t-\lambda} \leftarrow q_t$, then the current LSTM cell takes the summary vector f_t and the hidden state $h_{t-\lambda}$ as input. Moreover, this LSTM cell updates the cell state $c_{t-\lambda}$ into the cell state c_t, and generates the new hidden state h_t as output. As in [64], the LSTM cell used in our work has three gates: *forget gate* g_t, *input gate* i_t, and *output gate* o_t in addition to the hidden state h_t and the cell state c_t:

$$g_t = \text{Sigmoid}(\mathbf{W}_g[h_{t-\lambda}, f_t] + b_g) \tag{4.10}$$

$$i_t = \text{Sigmoid}(\mathbf{W}_i[h_{t-\lambda}, f_t] + b_i) \tag{4.11}$$

$$o_t = \text{Sigmoid}(\mathbf{W}_o[h_{t-\lambda}, f_t] + b_o) \tag{4.12}$$

$$\tilde{c}_t = \text{Tanh}(\mathbf{W}_c[h_{t-\lambda}, f_t] + b_c) \tag{4.13}$$

$$c_t = g_t \odot c_{t-\lambda} + i_t \odot \tilde{c}_t \tag{4.14}$$

$$h_t = o_t \odot \text{Tanh}(c_t) \tag{4.15}$$

4.3.5 Output Layer

The output vector h_t of the current LSTM cell is sent to a Sigmoid layer, which calculates the probability p_t of correctly answering the current question q_t by

$$p_t = \text{Sigmoid}(\mathbf{W}_2^T \cdot h_t + b_2). \tag{4.16}$$

4.3.6 Model Optimisation

To optimize the model, we use the cross-entropy loss function between the predicted probability of being correctly answered p_t and the true answer y_t. The following objective function is defined over training data:

$$\mathcal{L} = -\sum_t (y_t \log p_t + (1 - y_t) \log(1 - p_t)) \tag{4.17}$$

We initialise of the memory matrices ($\mathbf{M}_t^k$ and $\mathbf{M}_t^v$) and embedding matrices ($\mathbf{A}$ and $\mathbf{B}$) using a random Gaussian distribution $N(0, \sigma)$. While for weights and biases of the neural layers, we use Glorot uniform random initialization [46] for faster convergence. These randomly initialized parameters are optimized using the stochastic gradient descent (SGD) mechanism [18]. As we use Hop-LSTM at the sequence layer, during the backpropagation of the gradients, only the parameters of the connected LSTM cells (i.e., the ones responsible for the current prediction error) are updated. Other parameters, such as the embedding matrices, weight matrices, and bias vectors, are updated in each backpropagation iteration based on loss function values.

Note that, through training, the model can discover relevant latent concepts for each question and store their state values in the value matrix $\mathbf{M}^v$.

4.4 Experiments

In this section, we present the experiments of evaluating our proposed model SKVMN against the state-of-the-art KT models. These experiments aim to answer the following questions:

- Q1: What is the optimal size for a key-value memory (i.e., the key and value matrices) of SKVMN?
- Q2: How does SKVMN perform on predicting a student's answers to new questions, given an exercise history?
- Q3: How does SKVMN perform in discovering the correlation between latent concepts and questions?
- Q4: How does SKVMN perform on capturing the evolution of a student's knowledge states?

4.4.1 Datasets

We use five well-established datasets from the KT literature including Synthetic-5, ASSISTments2009, ASSISTments2015, Statics2011, and JunyiAcademy [78, 201, 137]. Table 4.1

summarizes the statistics of the data sets. A detailed description of these datasets is introduced in Chapter 3 (Section 3.3).

4.4.2 Baseline Methods

In order to evaluate the performance of our proposed model, we select the following three KT models as the baselines:

- Bayesian knowledge tracing (BKT) [31], which is based on Bayesian inference in which a knowledge state is modeled as a set of binary variables, each representing the understanding of a single concept.
- Deep knowledge tracing (DKT) [137] which uses recurrent neural networks to model student learning.
- Dynamic key-value memory networks (DKVMN) [201] which extends the memory-augmented neural networks (MANN) by a key-value memory and is considered as the state-of-the-art model for knowledge tracing.

Our proposed model is referred to as SKVMN in the experiments.

4.4.3 Metrics

We use the area under the Receiver Operating Characteristic (ROC) curve, referred to as AUC [97], to measure the prediction performance of the KT models. The AUC ranges in value from 0 to 1. An AUC score of 0.5 means random prediction (i.e. coin flipping). The higher an AUC score goes above 0.5, the more accurately a predictive model can perform.

4.4.4 Experimental Setup

We divided each dataset across students (i.e., no splitting across the same student sequence data) into 70% for training and validation and 30% for testing, except for Synthetic-5. This is because, as mentioned before, Synthetic-5 itself contains the training and test subsets of the same size. For each training and validation subset, we further divided it using the 5-fold cross-validation (e.g., 80% for training and 20% for validation). The validation subset was used to determine the optimal values for the hyperparameters, including the memory slot dimensions d_k for the key matrix and d_v for the value matrix.

We utilized the Adam optimizer [79] for SGD implementation with the momentum of 0.9 and learning rate γ of 0.01 annealed using a cosine function every 15 epochs for 120 epochs, then it remains fixed at 0.001. The LSTM gradients were clipped to improve the training [133]. For the other baselines, we follow the optimization procedures indicated in the original work for each of them [31, 201, 137].

A mini-batch of 32 is selected during the training for all datasets, except Synthetic-5, for which we use a mini-batch of 8 due to the relatively small number of training samples (i.e., exercises) in the dataset [12]. For each dataset, the training process is repeated five times, each time using a different initialization. We report the average test AUC and the standard deviation over these five runs.

Table 4.1: Dataset statistics

Dataset	#Questions	#Students	#Exercises	#Exercises per student
Synthetic-5	50	4,000	200,000	50
ASSISTments2009	110	4,151	325,637	78
ASSISTments2015	100	19,840	683,801	34
Statics2011	1,223	333	189,297	568
JunyiAcademy	575	199,549	25,628,935	128

Table 4.2: Comparison of SKVMN with DKVMN under different numbers of memory slots N and state dimensions d, where m refers to the number of parameters in each setting.

Dataset	d	N	SKVMN		DKVMN	
			AUC (%)	m	AUC (%)	m
Synthetic-5	10	50	83.11	15K	82.00	12k
	50	50	83.67	30k	82.66	25k
	100	50	**84.00**	57k	**82.73**	50k
	200	50	83.73	140k	82.71	130k
ASSISTments2009	10	10	**83.63**	7.8k	81.47	7k
	50	20	82.87	35k	**81.57**	31k
	100	10	82.72	71k	81.42	68k
	200	20	82.63	181k	81.37	177k
ASSISTments2015	10	20	**74.84**	16k	**72.68**	14k
	50	10	74.50	31k	72.66	29k
	100	50	74.24	66k	72.64	63k
	200	50	74.20	163k	72.53	153k
Statics2011	10	10	84.50	92.8k	82.72	92k
	50	10	**84.85**	199k	**82.84**	197k
	100	10	84.70	342k	82.71	338k
	200	10	84.76	653k	82.70	649k
JunyiAcademy	10	20	82.50	16k	79.63	14k
	50	10	82.41	31k	79.48	29k
	100	50	**82.67**	66k	79.54	63k
	200	50	82.32	163k	**80.27**	153k

Table 4.3: The AUC results of the four models BKT, DKT, DKVMN, and SKVMN over all datasets.

Dataset	BKT	DKT	DKVMN	SKVMN
Synthetic-5	62.0 ± 0.02	80.3 ± 0.1	82.7 ± 0.1	$\mathbf{84.0 \pm 0.04}$
ASSISTments2009	63.1 ± 0.01	80.5 ± 0.2	81.6 ± 0.1	$\mathbf{83.6 \pm 0.06}$
ASSISTments2015	64.2 ± 0.03	72.5 ± 0.1	72.7 ± 0.1	$\mathbf{74.8 \pm 0.07}$
Statics2011	73.0 ± 0.01	80.2 ± 0.2	82.8 ± 0.1	$\mathbf{84.9 \pm 0.06}$
JunyiAcademy	65.0 ± 0.02	79.2 ± 0.1	80.3 ± 0.4	$\mathbf{82.7 \pm 0.01}$

For the Triangular layer, the hyper-parameter values (a, b, c) of each triangular membership function are set based on the empirical analysis of each dataset.

4.5 Results and Discussion

In this section, we present the results and discuss our observations from the results.

4.5.1 Hyperparameters

To explore how the sizes of the key and value matrices can affect the model performance, we have conducted experiments to compare SKVMN with DKVMN under different numbers of memory slots N and state dimensions d, where $d = d_k = d_v$ so as to be consistent with the previous work [201]. In order to allow a fair comparison between the models SKVMN and DKVMN, we select the same set of state dimensions (i.e., $d = 10, 50, 100, 200$) and the same corresponding numbers of memory slots N on the datasets Synthetic-5, ASSISTments2009, ASSISTments2015, and Statics2011 to report the AUC results, following the settings originally reported in [201]. For the dataset JunyiAcademy, it was not considered in the previous work [201]. Considering that JunyiAcademy has the largest number of students and exercises among all datasets, we use the same settings for the numbers of memory slots and state dimensions as the ones for the second largest dataset ASSISTments2015. Table 4.2 presents the AUC results for all five datasets.

As shown in Table 4.2, compared with DKVMN, our model SKVMN can produce better AUC results with comparable parameters on the datasets Synthetic-5, ASSISTments2015, and Statics2011, and with fewer parameters on the datasets ASSISTments2009 and JunyiAcademy. Particularly, for the dataset ASSISTments2009, SKVMN yields an AUC of 83.63% with N=10, d=10, and m=7.8k, whereas DKVMN yields an AUC at 81.57% with N=20, d=50, and m=31k (nearly 4 times of 7.8k). Similarly, for the JunyiAcademy dataset, SKVMN yields an AUC of 82.67% with N=50, d=100, and m=66k, whereas DKVMN yields an AUC of 80.27% with N=50, d=200, and m=153k (more than twice of 66k).

Note that the optimal value of N for ASSISTments2015 is higher than the one for its previous version (i.e., ASSISTments2009). This implies that the number of latent concepts N increases in ASSISTments2015 in comparison to ASSISTments2009. Moreover, the optimal value of d generally reflects the complexity of the exercises in a dataset, and the dataset JunyiAcademy has exercises of higher complexity than other real-world datasets.

4.5.2 Prediction Performance

We have conducted experiments on comparing the AUC results of our model SKVMN with the other three KT models: BKT, DKT, and DKVMN. Table 4.3 presents the AUC results of all the models. It can be seen that our model SKVMN outperformed the other models over all five datasets. Particularly, the SKVMN model achieved an average AUC value that is at least 2% higher than the state-of-art model DKVMN on all real-world datasets ASSISTments2009, ASSISTments2015, Statics2011, and JunyiAcademy. Even for the only synthetic dataset (i.e., Synthetic-5), the SKVMN model achieved an average AUC value of 84.0 ± 0.04, in comparison with 82.7 ± 0.1 achieved by DKVMN. Note that the AUC values on ASSISTments2015 are the lowest among all datasets, regardless of the KT models. This reflects the difficulty of the KT task in this dataset due to its lowest exercise per student ratio, which not only makes the training process more difficult but also limits the effective use of sequence information to enhance the prediction performance. Figure 4.3 illustrates the ROC curves of these four models for each dataset.

In a nutshell, based on the AUC results in Table 4.3, we have the following observations. First, the neural models generally performed better than the Bayesian inference model (i.e., BKT). This is due to the power of these models in learning complex student learning patterns without the need to oversimplify the problem's assumption to keep it within the tractable computation limits, as the case in BKT. Second, the memory-augmented models DKVMN and SKVMN performed better than the DKT model, which does not utilize an external memory structure. This has empirically verified the effectiveness of external memory structures in storing past learning experiences of students, as well as facilitating the access of relevant information to enhance prediction performance. Third, the use of sequential dependencies among exercises in our SKVMN model enhanced the prediction accuracy in comparison to the DKVMN model, which primarily considers the latest observed exercise.

4.5.3 Analysis of Clustering Questions

To provide insights on how our proposed model SKVMN can correlate questions to their latent concepts, In Figure 4.4. (a)-4.4.(b), we present the clustering results of questions based on their correlated concepts in the dataset ASSISTments2009, generated by using DKVMN and SKVMN, respectively. This dataset was selected for two reasons. First, it has a reasonable number of questions (i.e., 110), enabling the visualization of clusters to be readable. Second, each question in this dataset is provided with a description as depicted in Figure 4.5, which is useful for validating how well the model discovers correlations between questions and their latent concepts.

As shown in Figure 4.4, both DKVMN and SKVMN discover that there are 10 latent concepts relating to the 110 questions in the dataset ASSISTments2009, where all questions in one cluster relate to common latent concepts and are labeled using the same color. It can be noticed that SKVMN performs significantly better than DKVMN since the overlapping between different clusters in SKVMN is smaller than in DKVMN. For example, in Figure 4.4. (a), question 105 is about the curve slope concept, which is close to the cluster for geometric concepts in brown color, while it is placed in the cluster for equation system concepts in blue color. Similarly, other overlaps can be observed, such as questions 38, 73, and 26. This

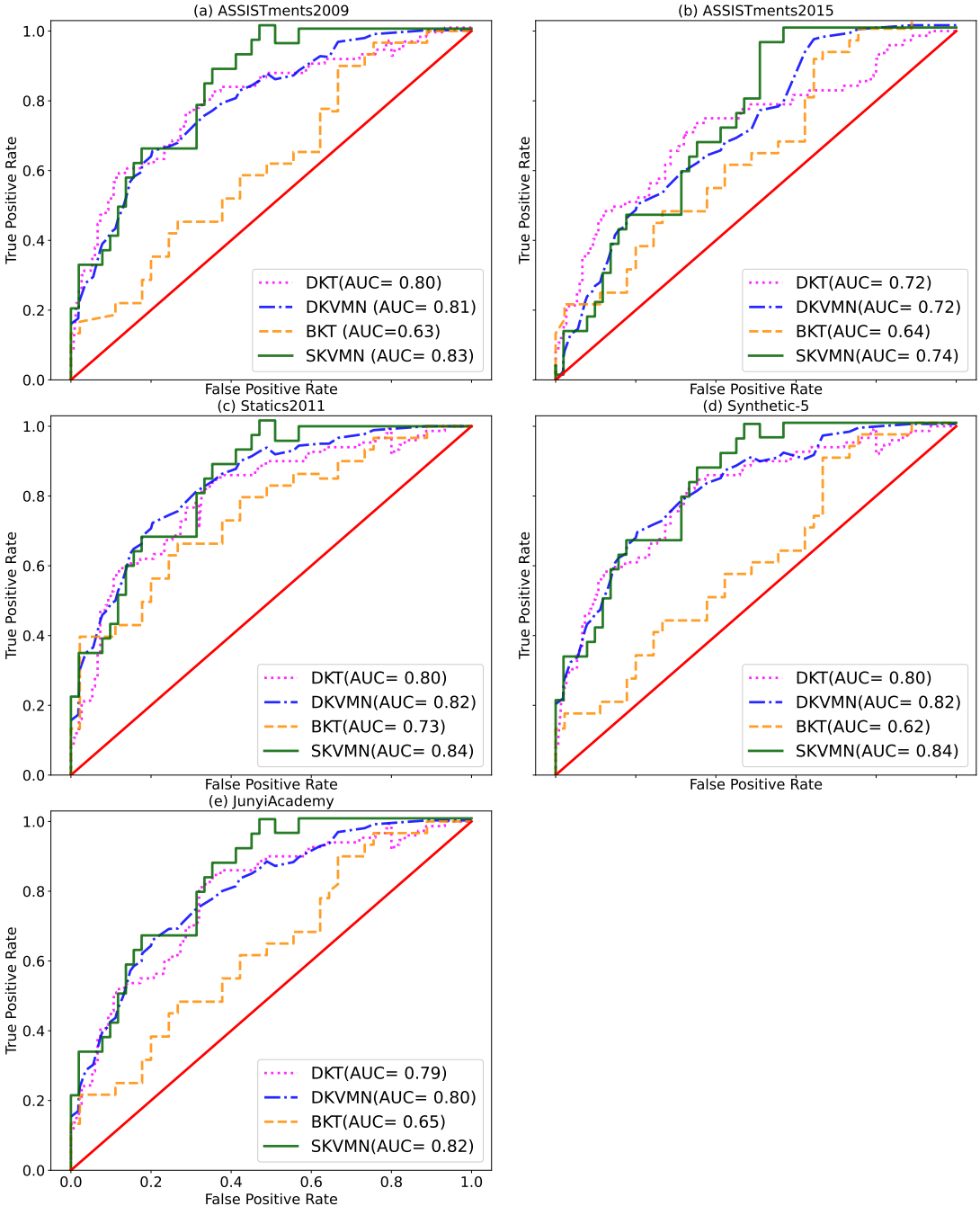


Figure 4.3: The ROC curve results of the four models BKT, DKT, DKVMN, and SKVMN over five datasets: (a) ASSISTments2009, (b) ASSISTments2015, (c) Statics2011, (d) Synthetic-5, and (e) JunyiAcademy.

indicates the effectiveness of SKVMN in discovering latent concepts as well as discovering questions that relate to these latent concepts.

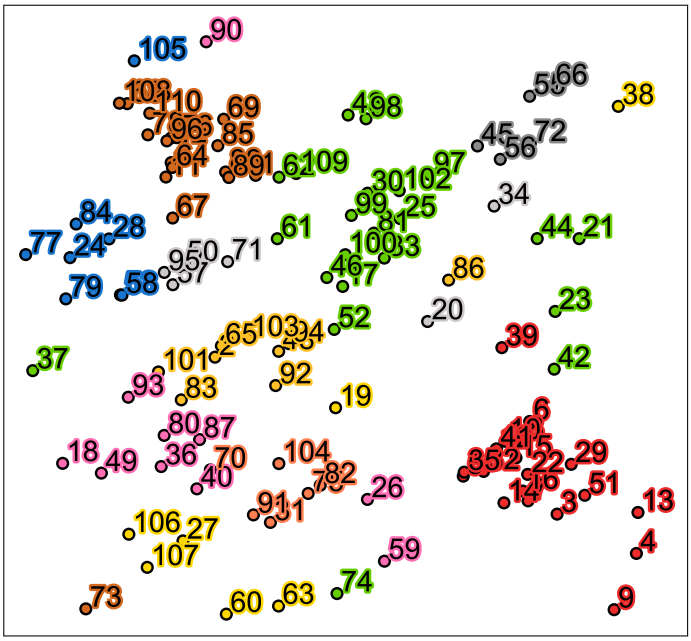
We can further verify the effectiveness of SKVMN in discovering latent concepts for questions using the question descriptions in Figure 4.5. For example, in Figure 4.4. (b), the questions 13, 19, and 30 fall in the same cluster in pink (top right corner). Their provided descriptions are "*Equivalent Fractions*","*Multiplication Fractions*", and "*Ordering Fractions*", respectively, which are all relevant to fractions concepts. Similarly, the questions 1, 81, and 92 have the descriptions "*Area Trapezoid*", "*Area Rectangle*", and "*Rotations*", respectively. These questions fall in the same cluster in light blue (bottom right corner) because they are about geometric concepts, such as area functions and transformations.

Note that SKVMN depends on identity vectors to aggregate questions with common concepts together. While the DKVMN depends on the attention vectors to perform this aggregation.

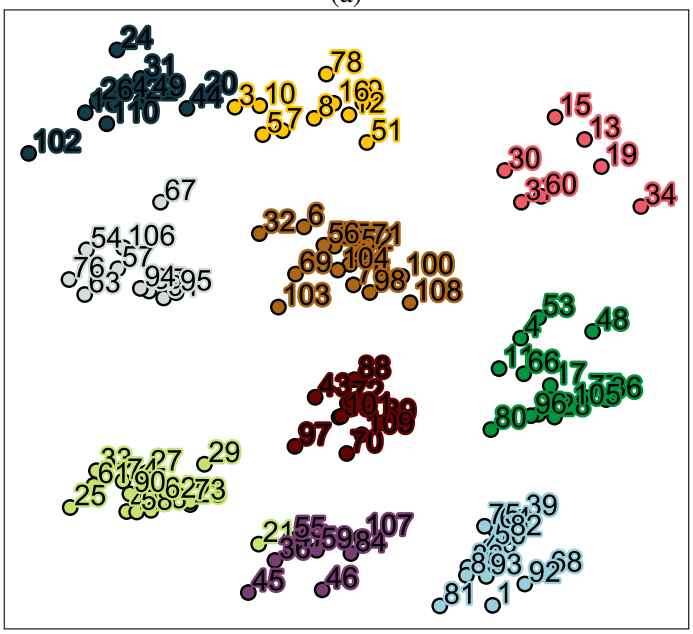
4.5.4 Evaluating Knowledge States

As previously discussed in Section 4.1, the knowledge states of a student may evolve over time through learning from a sequence of exercises. In order to illustrate this evolution process, Figure 4.1 shows a student's knowledge states over a sequence of 50 exercises from the ASSISTments2009 dataset. At each time step, a knowledge state consists of the concept states of five concepts $\{c_1, \ldots, c_5\}$, which are stored in the value matrix of a key-value memory augmented with DKVMN or SKVMN. Figure 4.1.(a) shows this student's knowledge states captured by DKVMN, while Figure 4.1. (b) shows this student's knowledge states captured by SKVMN.

In SKVMN, relevant questions are identified as shown in Figure 4.5. Comparing Figure 4.1. (a) and Figure 4.1. (b), it can be visually noticed that SKVMN has smoother updates to the concept states in the value matrix than DKVMN. For example, considering questions 23, 33, and 61, the student answered the first two questions incorrectly and the last one correctly, which result in a sudden update to the value of c_3 (i.e., the concept state of c_3) in the value matrix of DKVMN but a smoother update to the concept state of c^3 in the value matrix of SKVMN. Another example is questions 70 and 88, which correlate to the same latent concepts; the student answered the first one incorrectly and the second one correctly, which resulted in a significant update to the concept state of c_3 in the DKVMN's memory around indices 18, 19 and 20, while SKVMN's concept state of c_3 decreased in a smoother manner. This means that SKVMN considers the past performance of the student relevant to this concept. At its core, these differences in capturing concept states are due to the fact that DKVMN's write process only takes the question and the answer to calculate the erase and add vectors so that the knowledge state of DKVMN is biased to the most recently observed question. SKVNM has resolved this issue by considering the summary vector (i.e., current knowledge state and difficulty level of the current question) for the write process.



(a)



(b)

Figure 4.4: Clustering results of questions in the dataset ASSISTments2009 using: (a) DKVMN, and (b) SKVMN, where questions in the same color are correlating to the same latent concept.

Figure 4.5: Question descriptions in dataset ASSISTments2009, where the questions are clustered according to latent concepts being discovered by SKVMN.

21 Multiplication and Division Integers 23 Absolute Value 25 Subtraction Whole Numbers 27 Order of Operations +,-,/,* () positive reals 29 Counting Methods 33 Ordering Integers 38 Rounding 41 Finding Percents 58 Solving for a variable 61 Estimation 62 Ordering Real Numbers 73 Prime Number 74 Multiplication and Division Positive Decimals 90 Multiplication Whole Numbers	1 Area Trapezoid 39 Volume Rectangular Prism 40 Order of Operations All 50 Pythagorean Theorem 64 Surface Area Rectangular Prism 68 Area Circle 75 Volume Sphere 81 Area Rectangle 82 Area Triangle 83 Area Parallelogram 85 Surface Area Cylinder 92 Rotations 93 Reflection	6 Stem and Leaf Plot 32 Box and Whisker 35 Percent Of 42 Pattern Finding 52 Congruence 56 Reading a Ruler or Scale 69 Least Common Multiple 71Angles on Parallel Lines Cut by a Transversal 79 Solving Inequalities 98 Intercept 100 Slope 103 Recognize Linear Pattern 104 Simplifying Expressions positive exponents 108 Recognize Quadratic Pattern
43 Write Linear Equation from Situation 70 Equation Solving More Than Two Steps 72 Write Linear Equation from Ordered Pairs 88 Solving Systems of Linear Equations 89 Solving Systems of Linear Equations by Graphing 97 Choose an Equation from Given Information 99 Linear Equations 109 Finding Slope From Equation 101 Angles - Obtuse, Acute, and Right	54 Interior Angles Triangle 57 Perimeter of a Polygon 63 Scale Factor 65 Scientific Notation 67 Percents 76 Computation with Real Numbers 87 Greatest Common Factor 91 Polynomial Factors 94 Translations 95 Midpoint 106 Finding Slope From Situation	2 Area Irregular Figure 4 Table 11 Histogram as Table or Graph 17 Scatter Plot 28 Calculations with Similar Figures 48 Nets of 3D Figures 53 Interior Angles Figures with More than 3 Sides 66 Write Linear Equation from Graph 77 Number Line 80 Unit Conversion Within a System 86 Volume Cylinder 96 Interpreting Coordinate Graphs 105 Finding Slope from Ordered Pairs
14 Proportion 18 Addition and Subtraction Positive Decimals 22 Addition Whole Numbers 20 Addition and Subtraction Integers 24 Addition and Subtraction Fractions 26 Equation Solving Two or Fewer Steps 31 Circumference 44 Square Root 49 Complementary and Supplementary Angles 102 Distributive Property 110 Quadratic Formula to Solve Quadratic Equation		3 Probability of Two Distinct Events 5 Median 7 Mode 8 Mean 9 Range 10 Venn Diagram 12 Circle Graph 16 Probability of a Single Event 51 D.4.8-understanding-concept-of-probabilities 78 Rate
36 Unit Rate 45 Algebraic Simplification 46 Algebraic Solving 47 Percent Discount 55 Divisibility Rules 59 Exponents 84 Effect of Changing Dimensions of a Shape Prportionally 107 Parts of a Polyomial, Terms, Coefficient, Monomial, Exponent, Variable +		13 Equivalent Fractions 15 Fraction Of 19 Multiplication Fractions 30 Ordering Fractions 34 Conversion of Fraction Decimals Percents 37 Ordering Positive Decimals 60 Division Fractions

4.6 Summary

In this chapter, we introduced a novel model called Sequential Key-Value Memory Networks (SKVMN) for knowledge tracing. The SKVMN model aimed at overcoming the limitations of the existing KT models in modeling long-term dependencies in the past exercise answering sequence by learning a parametric membership function that identifies a subsequence of relevant answered exercises for a given input question. This subsequence enables the KT model to predict the probability of correctly answering an input question; thus, it better captures long-term dependencies related to the input question during the knowledge state estimation. SKVMN achieves this objective by combining a key-value memory with a Hop-LSTM module and optimizing them in an end-to-end manner. To evaluate the performance of the SKVMN model, we performed a comprehensive experimental analysis using five KT datasets. The experimental results showed that our proposed model outperforms comparative KT models over all datasets. Despite the potential of the SKVMN model, it has some limitations, including the lack of explicit representation for learning concept relationships and ignoring the student's forgetting behavior (see Section 2.2) during knowledge state estimation.

Chapter 5

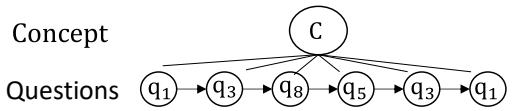
Deep Graph Memory Networks for Forgetting-Robust Knowledge Tracing

5.1 Overview

In the real world, students tend to forget or remember latent concepts rather than questions themselves, and the relationships across these latent concepts would have a forgetting effect on a student's knowledge states. For example, a student might be able to correctly answer a new math question which she has never seen before after she has mastered the latent concepts (e.g., dot product) behind such a question. Figure 5.1 illustrates two different forgetting modeling paradigms: *over one latent concept shared across questions* versus *over multiple latent concepts shared across questions*. Let us consider two forgetting features [121]: (i) *Time Lapse* that counts the number of time steps lapsed since a question has been answered last time, and (ii) *Trials* that refers to the total number of times that a student has answered a question. In Figure 5.1.a, forgetting is modeled over one latent concept shared by all the questions that are identified by their question tags, while in Figure 5.1.b, forgetting is modeled over multiple concepts that are identified by their concept tags and shared by the questions. We can see that the question q_1 has appeared twice in this sequence. At the time step t_6, q_1 thus has the forgetting features *Time Lapse*=5 and *Trials*=2 in Figure 5.1.a and *Time Lapse*=1 and *Trials*=6 in Figure 5.1.b. Accordingly, forgetting is likely to occur on q_1 in Figure 5.1.a. However, since the concept underlying q_1 is c_2, which indeed underlies all the questions in this sequence, the student has learned c_2 through all these questions, and forgetting would unlikely happen at the time step t_6. Therefore, modeling forgetting features over multiple concepts can improve the tracing ability of a student's knowledge states, whereas modeling over one concept cannot achieve this outcome.

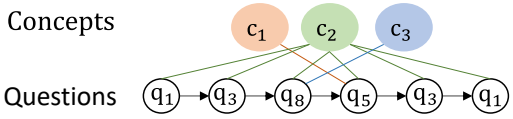
Nonetheless, when learning forgetting features over a concept space, a difficulty arises: *since only a question answering sequence is available, how can we model forgetting features effectively in terms of latent concepts behind questions rather than questions themselves?* We observe that the key to tackling this difficulty is to design a KT model that can not only learn latent concepts behind questions via a question answering sequence but also capture the relationships among latent concepts that are mastered by a student. More importantly, such a KT model should have the capability of *dynamically* tracing changes in latent concepts and their relationships in light of a student's performance over time. Even with modeling forgetting fea-

(a) Forgetting Over One Latent Concept



Time Step	t_1	t_2	t_3	t_4	t_5	t_6
Time Lapse	0	0	0	0	3	5
Trials	1	1	1	1	2	2

(b) Forgetting Over Multiple Latent Concepts



Time Step	t_1	t_2	t_3		t_4		t_5	t_6
Concepts	c_2	c_2	c_2	c_3	c_1	c_2	c_2	c_2
Time Lapse	0	1	1	0	0	1	1	1
Trials	1	2	3	1	1	4	5	6

Figure 5.1: An example for illustrating the difference between modeling forgetting over one latent concept and over multiple latent concepts, where *Time Lapse* and *Trials* are two forgetting features used for modeling forgetting.

tures over a concept space upon a deep graph memory structure, there is still one question to be answered: *how can we effectively use forgetting features to reflect a student's forgetting behavior (i.e., a decline of memory retention in time) and count for its effect on predicting answers?* This poses a need to find an effective way to combine forgetting features with knowledge states (i.e., skill mastery information).

In this chapter, we introduce a novel forgetting-robust KT model, namely *Deep Graph Memory Network* (DGMN), that answers the two previously identified questions by addressing the following challenges:

(1) Cognitive modeling studies on human memory have reported a knowledge decline trend over time due to forgetting behaviors, known as the *forget curve theory* [39]. However, it is challenging to model such forgetting behaviors into knowledge states and reflect them in a KT model for predicting a student's learning performance. Thus, the following questions arise: How to represent the decline patterns in a student's memory over time? How can such representations be incorporated during prediction for a knowledge tracing model?

(2) Concept learning is a phenomenal ability of human beings, which involves extracting latent concepts from questions, i.e., concepts underlying questions that are semantically

similar or related, such as addition, subtraction, and division in a math course, and identify their relationships [123]. However, it is not yet clear how to effectively leverage information of latent concepts related to an observed question answering sequence for predicting a student's learning progress in a KT model.

The novelty of this proposed model comes into three folds: (1) modeling forgetting features over multiple latent concepts in a concept space rather than over one latent concept shared across all questions; (2) learning a dynamical latent concept graph to reflect the dynamics of a student's knowledge states over latent concepts and their relationships; (3) incorporating a forget gating mechanism to account for the effect of forgetting features on a student's learning performance.

In summary, the main contributions of this chapter are summarized as follows:

- We propose a novel KT model, namely DGMN, which is augmented with a deep graph memory structure to dynamically trace a student's knowledge states over latent concepts in an attentive key-value memory and the relationships among latent concepts in a latent concept graph.
- We design the modeling of forgetting features over a latent concept space considering their mutual relationships and devise a forget gating mechanism to combine forgetting features with knowledge state information in our proposed DGMN model for predicting a student's learning performance.
- We propose to capture embedding representations of latent concepts and their relationships in a dynamic latent concept graph and leverage its useful information to rank questions for improving model optimization.
- We have extensively evaluated the performance of our proposed DGMN model by comparing it against the state-of-the-art deep learning and graph KT models on four well-established KT datasets.

The remainder of this chapter is organized as follows. Section 5.2 defines the research problem. Section 5.3 describes our proposed KT method. Section 5.4 presents the experimental design. Section 5.5 discusses the experimental results and findings. Section 5.6 concludes the chapter.

5.2 Problem Definition

In this section, we extend the general definition of the knowledge tracing problem introduced in Section 1.1.2 to formally consider the impact of a student's forgetting behavior on the estimated knowledge state. Assuming a knowledge state estimator model $\varphi_\tau : \mathrm{q_t}, \mathrm{X} \rightarrow \hat{\mathrm{S}}_\mathrm{t}$ with parameters τ, we count for a student's forgetting behavior impact by defining the estimated knowledge state $\hat{\mathrm{S}}_\mathrm{t}$ as being mixed with a noise component ϵ_t resulting from the forgetting behavior. That is $\hat{\mathrm{S}}_\mathrm{t} = \mathrm{S_t} + \epsilon_t$, where $\mathrm{S_t}$ is the true knowledge state at time point t. Thus, we aim to learn a forgetting gate $\mathrm{fg}_\omega : \hat{\mathrm{S}}_\mathrm{t}, \mathrm{C}^{\mathrm{q_t}} \rightarrow \mathrm{S_t}$ that filters the noise component parameterized with ω and conditioned on $\mathrm{C}^{\mathrm{q_t}}$ as the set of relevant learning concepts to question $\mathrm{q_t}$. Moreover, we assume

that relationships between existing learning concepts in the task are captured using a concept graph $\mathrm{G}(\mathrm{V}, \mathrm{E})$, where V is the set of node embeddings, each representing a learning concept, and E is the set of edges representing their mutual relationships. To consider such relationships among learning concepts, we condition on G and S_t during answer prediction. Thus, our KT hypothesis function is defined as $f_\theta : (S_t, G) \rightarrow \hat{y}_t$ and parameterized with θ. We optimize the parameters τ, ω, θ by minimizing an answer prediction loss function $\mathcal{L}$ as follows:

$$\underset{\theta^* \in \Theta, \omega^* \in \Omega, \tau^* \in T}{\arg\min} \overbrace{\mathcal{L}(\underbrace{f_\theta(\mathrm{fg}_\omega(\underbrace{\varphi_\tau(\mathrm{q_t}, \mathrm{X})}_{\text{Knowledge Estimate}}, C^{q_t})}_{\text{Forget Gate}}), G)}^{\text{Answer Prediction}}, y_t) \tag{5.1}$$

5.3 Methodology

In this section, we introduce the details of different building blocks in our proposed model named *Deep Graph Memory Network* (DGMN) for forgetting-robust knowledge tracing.

5.3.1 Model Overview

The DGMN model has three main components: *attention memory* (AM), *latent concept graph* (LCG), and *answer prediction*. The AM component stores a multi-dimensional representation of a student's knowledge state over the involved latent concepts (e.g., math topics) in a learning task. Thus, it enables to trace the knowledge progress for each concept through its dedicated dimension in knowledge states. The linkage between answered questions and latent concepts is done through an attentive memory read mechanism that we will detail in this section. To address the challenge of leveraging relationship data among latent concepts, the LCG component utilizes graph representation learning to capture concept relational dependencies. The LCG builds upon representations learned by the AM component to construct a dynamic concept-to-concept graph and uses graph convolution [80] to distill a representation for latent concept dependencies. We count for a student's forgetting behavior using a novel gating mechanism that combines the knowledge state representation from the AM component, the concept dependencies representation from the LCG component, and forgetting features from the past answering sequence into a consolidated representation for answer prediction. Figure 5.2 shows the model architecture of DGMN.

5.3.2 Attention Memory

The purpose of this module is to learn embedding representations of the latent concepts in a KT task and to track the dynamics of a student's knowledge states.

5.3.2.1 Concept Embedding Memory

This memory stores an embedding vector (i.e., a memory slot) for each latent concept. It is represented as a matrix $\mathbf{M}^{\mathrm{k}} \in \mathbb{R}^{\mathrm{N} \times \mathrm{d_k}}$, where N is the number of latent concepts and d_k is the

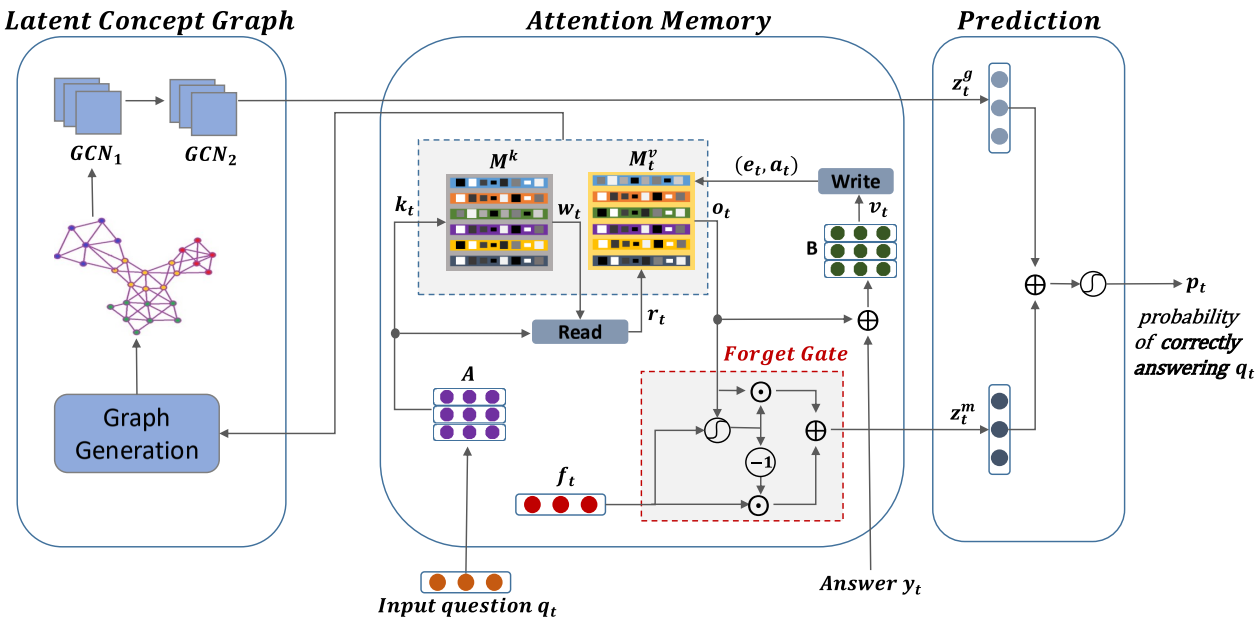


Figure 5.2: Model architecture of our proposed DGMN model, where the latent concept graph (LCG) module is on the left side, the attention memory (AM) module is in the middle, and the answer prediction layer is on the right side.

embedding dimension.

Given a one-hot encoding vector $\delta(q_t) \in \mathbb{R}^{|Q|}$ of question q_t, we multiply it with a learnable embedding matrix $\mathbf{A} \in \mathbb{R}^{|Q| \times d_k}$ to get a question embedding vector $k_t \in \mathbb{R}^{d_k}$. Then, the inner product between k_t and $\mathbf{M^k}$ is supplied to a Softmax layer to obtain a relevance vector $w_t \in \mathbb{R}^N$, as calculated by

$$w_t(i) = \text{Softmax}(k_t^T \mathbf{M}^k)(i) \tag{5.2}$$

where $\text{Softmax}(u_i) = e^{u_i} / \sum_j e^{u_j}$ and $\mathbf{M}^k(i)$ is the embedding vector for the i-th latent concept. This relevance vector w_t reflects the relevance between the current question q_t and each latent concept embedding stored in $\mathbf{M}^k$. Thus, it is then used to read a student's knowledge state from the concept of state memory.

5.3.2.2 Concept State Memory

This memory holds the current concept states of a student, represented as a matrix $\mathbf{M}_t^v \in \mathbb{R}^{N \times d_v}$, where N is the number of latent concepts and d_v is the state dimension.

Given a relevance vector w_t, we acquire a vector $r_t \in \mathbb{R}^{d_v}$ from the $\mathbf{M}_t^v$ memory using a weighted sum across memory slots as

$$r_t = \sum_{i=1}^{N} w_t(i) \mathbf{M}_t^v(i). \tag{5.3}$$

Here, r_t indicates the concept state information from $\mathbf{M}_t^v$ relevant to the input question, yet it does not include information about the question itself. To resolve this, we linearly combine it with the question embedding vector $k_t \in \mathbb{R}^{d_k}$, and then feed them to a Tanh layer to calculate a concept summary vector $o_t \in \mathbb{R}^N$ as

$$o_t = \text{Tanh}(\mathbf{W} [r_t, k_t] + b), \tag{5.4}$$

where $\text{Tanh}(u_i) = (e^{u_i} - e^{-u_i}) / (e^{u_i} + e^{-u_i})$, $\mathbf{W} \in \mathbb{R}^{N \times (\mathbf{d_v} + \mathbf{d_k})}$ is a weight matrix, and b is a bias vector. The concept summary vector o_t reflects the information regarding the input question q_t and its relevant concept states in $\mathbf{M}_t^v$.

5.3.2.3 Forget Gating Mechanism

To model a student's forgetting behavior, we first need to identify questions that are relevant to an input question q_t in terms of its latent concepts. Let $C(q_t)$ be the set of latent concepts underlying a question q_t, i.e., for each latent concept $c_i \in C(q_t)$, the relevance weight $w_t(i)$ to $\mathbf{M}^k(i)$ is greater than a relevancy threshold τ. A question q_j previously occurring in $\mathbf{x}_t$ is *relevant* to c_i iff $c_i \in C(q_j)$. Thus, at each time step t, we obtain two forgetting features:

- *Time Lapse*: the time lapse $= t - t'$ where t' refers to the most recent time when a question in $\mathbf{x}_t$ relevant to the latent concept c_i was observed.
- *Trials*: the total number of trials for questions in $\mathbf{x}_t$ that are relevant to the latent concept c_i.

Calculating these forgetting features for N latent concepts at time step t gives us a matrix $\mathbf{F}_t \in \mathbb{R}^{N\times 2}$. To get a combined forgetting vector $f_t \in \mathbb{R}^N$, we flatten the matrix $\mathbf{F}_t$ into an input vector $f_{in} \in \mathbb{R}^{2N}$ and send it to a *Tanh* layer as follows:

$$f_t = \text{Tanh}(\mathbf{W} \cdot f_{in} + b). \tag{5.5}$$

where $\mathbf{W} \in \mathbb{R}^{N\times 2N}$ and b are learnable weight matrix and bias vector respectively.

To measure the importance of f_t to the input question q_t, we perform an element-wise multiplication with the relevance vector $w_t \in \mathbb{R}^N$ to obtain a forget summary vector $fs_t \in \mathbb{R}^N$ as

$$fs_t = w_t \odot f_t \tag{5.6}$$

Then, we use a forget gating mechanism to combine information from the concept summary vector o_t and the forget summary vector fs_t. The forget gating mechanism works in two steps. First, a gating vector $gw_t \in \mathbb{R}^N$ is generated by applying a Sigmoid function to a linear combination of vectors o_t and fs_t. Then, we calculate a combined memory vector $z_t^m \in \mathbb{R}^N$ based on the gating vector gw_t:

$$\begin{aligned} gw_t &= \text{Sigmoid}(\mathbf{W}_1 \cdot o_t + \mathbf{W}_2 \cdot fs_t); \\ z_t^m &= gw_t \odot o_t + (1 - gw_t) \odot fs_t, \end{aligned} \tag{5.7}$$

where $\mathbf{W}_1 \in \mathbb{R}^{N\times N}$ and $\mathbf{W}_2 \in \mathbb{R}^{N\times N}$ are learnable weight parameters, and $\odot$ indicates an element-wise multiplication. Later, z_t^m is used along with the information from a latent concept graph (will be introduced in Section 5.3.3) to predicate the probability of correctly answering q_t.

5.3.2.4 Memory Update Procedure

Updating the state memory $\mathbf{M}_t^v$ is controlled by two signals: *add signal* a_t and *erase signal* e_t. These signals decide the proportion of information being changed on $\mathbf{M}_t^v$ after observing a new question and its answer from the student. a_t decides new information be written to the memory, while e_t decides old information to be removed from the memory.

Suppose that $(\delta(q_t), y_t)$ is the latest observed question-answer tuple; we embedded it with an embedding matrix $\mathbf{B} \in \mathbb{R}^{2|Q|\times d_v}$ to obtain an embedding vector $v_t^q \in \mathbb{R}^{d_v}$. Then, we concatenate this embedding vector with the summary vector o_t to combine information on the latest question-answer along with the corresponding knowledge state summary. The concatenated update vector $v_t = [v_t^q, o_t] \in \mathbb{R}^{(d_v+N)}$ is used to update the $\mathbf{M}_t^v$ memory using erase and add signals [52]. The erase signal $e_t \in \mathbb{R}^{d_v}$ is obtained by the following equation:

$$e_t = \text{Sigmoid}(\mathbf{W}_e \cdot v_t + b_e). \tag{5.8}$$

where $\mathbf{W}_e \in \mathbb{R}^{d_v\times(d_v+N)}$ is a learnable weight matrix and b_e is a bias vector. Given the erase signal e_t, the concept state memory is updated from $\mathbf{M}_t^v$ to $\tilde{\mathbf{M}}_{t+1}^v$ as

$$\tilde{\mathbf{M}}_{t+1}^v(i) = \mathbf{M}_t^v(i)[1 - w_t(i)e_t]. \tag{5.9}$$

The add signal $a_t \in \mathbb{R}^{d_v}$ is obtained by the following equation:

$$a_t = \text{Tanh}(\mathbf{W}_a \cdot v_t + b_a). \tag{5.10}$$

where $\mathbf{W}_a \in \mathbb{R}^{d_v \times (d_v+N)}$ is a learnable weight matrix and b_a is a bias vector. Finally, the concept state memory $\mathbf{M}^v_{t+1}$ is calculated as

$$\mathbf{M}^v_{t+1}(i) = \tilde{\mathbf{M}}^v_{t+1}(i) + w_t(i)a_t. \tag{5.11}$$

5.3.3 Latent Concept Graph

Although the attention memory module captures the temporal dynamics of a student's knowledge state, it cannot represent relationships between latent concepts. For example, observing knowledge growth for a given latent concept would impact the prediction of answers to related latent concepts. Thus, we learn a graph representation to capture such relationships.

Given an attention memory ($\mathbf{M}^k$, $\mathbf{M}^v_t$), a *latent concept graph* (LCG) over ($\mathbf{M}^k$, $\mathbf{M}^v_t$) is a weighted graph $G = (V, E, \omega)$ where each vertex in V corresponds to a latent concept in $\mathbf{M}^k$ and each edge in E is weighted based on the similarity of two latent concepts it connects in $\mathbf{M}^v_t$. Specifically, we define $\omega : \mathbf{M}^v_t \times \mathbf{M}^v_t \rightarrow [0, 1]$ such that $\omega(\mathbf{M}^v_t(i), \mathbf{M}^v_t(j)) \geq \mu$, i.e., the similarity value for the edge between two latent concepts c_i and c_j is no less than μ, where $\mathbf{M}^v_t(i)$ and $\mathbf{M}^v_t(j)$ are the concept state vectors for c_i and c_j, respectively, and $\mu \in [0, 1]$ is a similarity threshold.

To extract useful information from a latent concept graph, we employ graph convolutional networks (GCNs) [38, 80]. This is achieved by applying convolution over the neighbors of each vertex in the graph. The input of a GCN layer includes 1) the latent concept embedding memory $\mathbf{M^k} \in \mathbb{R}^{N \times d_k}$ in our case, and 2) the graph adjacency matrix with self-loops $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\hat{\mathbf{A}} \in \mathbb{R}^{N \times N}$, where $\mathbf{I}$ is the identity matrix, and 3) the diagonal degree matrix $\hat{\mathbf{D}} \in \mathbb{R}^{N \times N}$ containing the degree information for each vertex in $\hat{\mathbf{A}}$. The propagation rule of a GCN layer is calculated through a convolution process [80]:

$$\mathbf{H}^{i+1} = ReLU(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^i\mathbf{W}^i) \tag{5.12}$$

where $\text{ReLU}(u) = max(0, u)$ is a non-linear activation function, $\mathbf{W}^i$ is a learnable weight matrix. $\mathbf{H}^i \in \mathbb{R}^{N \times d_k}$ is an input feature matrix which is the output of the previous GCN layer, and $\mathbf{H}^0 = \mathbf{M}^k$ for the first GCN layer.

Finally, we calculate the *graph summary vector* $z^g_t \in \mathbb{R}^N$, which attends the GCN output to the current question q_t using the relevance vector w_t

$$z^g_t = \text{Tanh}(\mathbf{W}(\sum_{i=1}^{N} w_t(i)\mathbf{H}(i))^\intercal + b), \tag{5.13}$$

where $\mathbf{W} \in \mathbb{R}^{N \times d_k}$ is a learnable weight matrix, $\mathbf{H}$ is the output matrix of the final GCN layer in our model, and b is a bias vector.

5.3.4 Answer Prediction

To predict the probability of correctly answering an input question q_t, we concatenate the combined memory vector z_t^m from the AM module with the graph summary vector z_t^g from the LCG module and feed them to a fully connected layer with Sigmoid activation. Then, we multiply the output logits vector with the one-hot encoding vector $\delta(q_t)$ of question q_t as

$$p_t = \text{Sigmoid}(\mathbf{W}\left[z_t^m, z_t^g\right] + b) \cdot \delta(q_t). \tag{5.14}$$

where $\mathbf{W} \in \mathbb{R}^{|Q|\times 2N}$ is a learnable weight matrix, b is a bias vector, and p_t is a scalar predicting the correct answer probability for q_t.

5.3.5 Model Optimization

Different questions may have different impacts on a student's knowledge state. We observe that the impact of a question largely depends on its latent concepts and the importance of these concepts in mastering a learning subject. Consider the dataset *Statics2011* for example, which was collected from an undergraduate mechanical engineering course (will be further discussed in Section 5.4.1). As "Forces" is a fundamental concept in the course, correctly answering questions about this concept can significantly help improve the performance on the other concepts.

Following this observation, we exploit a question ranking technique during model training. Given a training mini-batch $\hat{D}^{\text{train}=\{(q_j,y_j)\}_{j=1}^m}$, the ranks for questions in $\hat{D}^{\text{train}}$ are calculated as follows:

$$\gamma(q_j) = \sum_{i=1}^{N} w_j(i)\, \mathbf{d}(i,i) \tag{5.15}$$

where w_j is the relevance vector of q_j from the attention memory module, $\mathbf{d}(i,i)$ refers to the degree value of the i-th latent concept c_i from the latent concept graph, and N refers to the total number of latent concepts in the KT task.

Let $\Gamma = \{\gamma(q_j)|q_j \in \hat{D}^{\text{train}}\}$ be the set of question ranks for $\hat{D}^{\text{train}}$. Then, we apply min-max normalization to ensure that the ranks in Γ are normalized s.t. each question q_j is associated with a normalized rank $\overline{\gamma}(q_j) \in [0,1]$. Then, given a training mini-batch $\hat{D}^{\text{train}}$, our model aims to minimize the following objective function:

$$L = \frac{1}{|\hat{D}^{\text{train}}|}\sum_{i=1}^{|\hat{D}^{\text{train}}|} \overline{\gamma}(q_i) \cdot l_i. \tag{5.16}$$

where $l_i = -(y_i \log(p_i) + (1-y_i)\log(1-p_i))$ is the cross-entropy loss for predicting the answer y_i of question q_i.

The training process of our proposed DGMN model is described in Algorithm 1.

Algorithm 1 Training DGMN

Initialize:
Weights, bias parameters, and embedding matrices
Begin:
for epoch $\in 1, \ldots, n$ **do**
  for mini-batch $\in 1, \ldots, m$ **do**
    for all sample $\in$ mini-batch **do**
      Embed an input question q_t
      Get relevance vector w_t (Eq. 5.2)
      Get read vector r_t (Eq. 5.3)
      Get concept summary vector o_t (Eq. 5.4)
      Get forgetting summary vector fs_t (Eg. 5.6)
      Get combined memory vector z^m_t (Eq. 5.7)
      Get graph summary vector $z^g{}_t$ (Eq. 5.13)
      Compute the probability p_t (Eq.5.14)
      Perform update on attention memory (Eq. 5.11)
    end for
    Get question ranking in the mini-batch (Eq. 5.15)
    Perform Adam [148] (Eq. 5.16)
    Update the LCG graph
  end for
end for

5.4 Experiments

In this section, we present the experimental design for evaluating our proposed DGMN model, aiming to answer the following questions:

Q1: How does our proposed DGMN model perform against the state-of-the-art KT models?

Q2: How different components (i.e., forget gating mechanism, latent concept graph and question ranking) in the proposed DGMN model affect on its performance?

Q3: How effectively can a learned latent concept graph reflect latent concepts and their relationships?

Q4: How does modeling forgetting over multiple concepts in our model compared to other deep learning models assuming only a single concept?

5.4.1 Datasets

In this section, we use in our experiments four datasets from the KT literature including ASSISTments2009, Statics2011, Synthetic-5, and Kddcup2010. Table 5.1 summarizes some statistics for each dataset. A detailed description of these datasets is introduced in Chapter 3 (Section 3.3).

Table 5.1: Statistics of datasets

Dataset	#Students	#Questions	#Exercises	#Concepts
ASSISTments2009	4,151	110	325,637	110
Statics2011	335	1,362	190,923	85
Synthetic-5	4,000	50	200,000	5
Kddcup2010	575	436	607,026	112

5.4.2 Baseline Methods

This experiment answers the question *Q1*. This is done by comparing the performance of our DGMN model with the state-of-the-art KT models on four well-established datasets in the KT literature. The KT models for comparison include:

- **DKT+forget** [121]: This model extends the DKT model [137] by adding forgetting features calculated from a question answering sequence.
- **DKVMN** [201]: This model uses a key-value memory to track the knowledge state of a student across latent concepts.
- **GKT** [122]: This model uses a graph neural network to extract information that captures dependencies across questions.
- **SAKT** [126]: This model follows an attention mechanism [7] to weigh the importance of previously answered questions in a sequence for predicting the latest one.
- **AKT** [45]: This model combines an attention model with Rasch model-based embeddings, which exponentially decays attention weights w.r.t. the distance of questions in a sequence in order to account for student's forgetting effect.
- **HawkesKT** [180]: This model uses the *Hawkes* process [194] to count for a student's forgetting behavior. It adds temporal excitement coefficients for questions in the past answer history and correlates these coefficients with the latest question to model mutual excitation that captures cross-skill effects.
- **SAINT+** [157]: This model follows an encoder-decoder transformer architecture [177] to model the sequence of past question answering. Moreover, it incorporates auxiliary temporal features such as elapsed time for answering a question and the time gap between two adjacent answer attempts to count for a student's forgetting behavior.

For hyperparameters of the baseline methods, we use the best configuration as suggested by the original work in the case that the dataset was evaluated. If the dataset was not present in the original work, we perform hyperparameter tuning using 5-fold cross-validation and use the best configuration during the comparison. For all the models including our DGMN model, we execute five independent runs and report the averaged results. We also perform the statistical significance *t-test* on the results with $p\text{-}value = 0.05$.

5.4.3 Metrics

We use the area under the Receiver Operating Characteristic (ROC) curve, referred to as AUC [97], to measure the prediction performance of the KT models. The AUC ranges in value from 0 to 1. An AUC score of 0.5 means random prediction (i.e. coin flipping). The higher an AUC score goes above 0.5, the more accurately a predictive model can perform.

5.4.4 Experimental Setup

For each dataset, we divide it into training and testing sets using a split ratio of $(70\%, 30\%)$ for training and testing, respectively. We use a 5-fold cross-validation strategy in our experiments. The attention memory matrices ($\mathbf{M}^{k}$ and $\mathbf{M}_{t}^{v}$) and embedding matrices ($\mathbf{A}$ and $\mathbf{B}$) are initialized using a zero-mean random Gaussian distribution $N(0, \sigma)$. For parameters of the neural layers, we use *Glorot* uniform random initialization [46] for faster convergence. An empirical evaluation on validation data (i.e., 20% of training data) is conducted to determine the hyper-parameters of our model, including the memory slot dimensions $d_k = 50$ and $d_v = 100$, and the relevancy threshold $\tau = 0.8$ for forgetting features calculation, the similarity function ω (*cosine similarity* [124]), and the similarity threshold $\mu = 0.25$. We optimize the model's parameters using *Adam* gradient decent algorithm [148].

5.5 Results and Discussion

In this section, we present the results of our experiments and discuss the findings from these experiments.

5.5.1 Prediction Performance

This experiment compares the performance of the DGMN model with the state-of-the-art KT models on four KT datasets. Table 5.2 reports the average AUC results and Figure 5.3 depicts average AUC values using bar plots.

We observe that DGMN outperforms all the other models on all the datasets and achieves an AUC improvement over the nearest model (i.e. SAINT+) by a margin of 1.7%, 1.3%, 2.1%, and 1.8% on the datasets *ASSISTments2009*, *Statics2011*, *Synthetic-5*, and *Kddcup2010*, respectively. These results are statistically significant, i.e., *p-value*<0.05, based on the *t-test* conducted in different runs. Notice that, DGMN and AKT both obtain the lowest AUC values on *Kddcup2010* among all the databases. This is due to the complexity that this dataset adds to the KT benchmarks as it has the largest number of exercises and latent concepts among all the datasets.

The results highlight the performance advantage of DGMN by utilizing graph features from LCGs and forgetting features from attention memory, in comparison to only forgetting features as in DKT-forget, only a key-value memory without any forgetting features as in DKVMN, and only graph features as in GKT.

Table 5.3 summarizes the training time complexity and the memory complexity for each model.

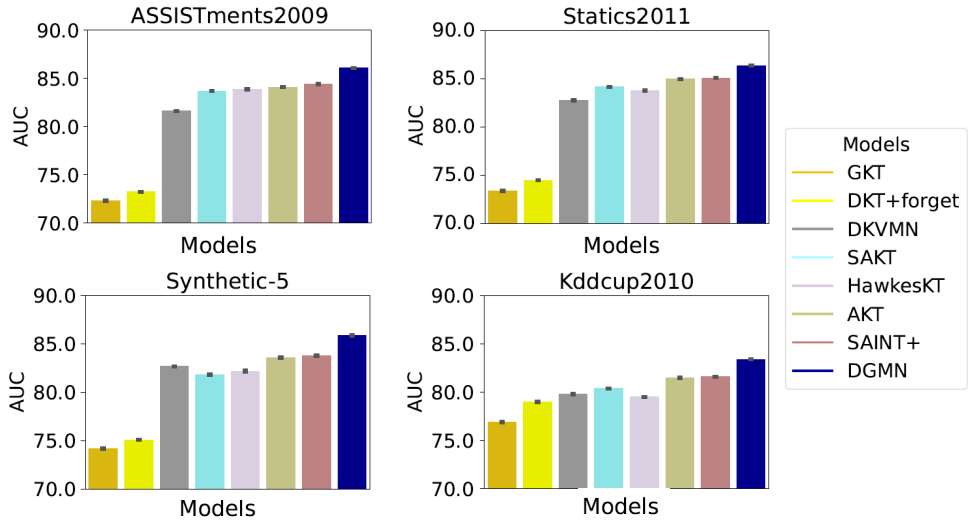


Figure 5.3: The average AUC values comparing DGMN with the state-of-the-art KT models over all the datasets (best in colors).

Table 5.2: The average AUC results for comparing DGMN with the state-of-the-art KT models over all the datasets.

Model	Forget	Graph	Rank	Knowledge Representation Model	Dataset			
					ASSISTments2009	Statics2011	Synthetic-5	Kddcup2010
GKT [122]	×	✓	×	GNN	72.3 ± 0.02	73.4 ± 0.03	74.2 ± 0.01	76.9 ± 0.01
DKT+forget [121]	✓	×	×	RNN	73.2 ± 0.02	74.5 ± 0.01	75.1 ± 0.03	79.0 ± 0.01
DKVMN [201]	×	×	×	Key-value memory	81.6 ± 0.03	82.8 ± 0.02	82.7 ± 0.01	79.8 ± 0.02
SAKT [126]	×	×	×	Attention mechanism	83.7 ± 0.02	84.2 ± 0.01	81.9 ± 0.03	80.4 ± 0.02
HawkesKT [180]	✓	×	×	Factorization machines	83.9 ± 0.04	83.8 ± 0.05	82.2 ± 0.05	79.5 ± 0.03
AKT [45]	✓	×	×	Attention mechanism	84.1 ± 0.03	85.0 ± 0.02	83.6 ± 0.03	81.5 ± 0.02
SAINT+ [157]	✓	×	×	Attention mechanism	84.4 ± 0.04	85.1 ± 0.03	83.8 ± 0.03	81.6 ± 0.05
DGMN (ours)	✓	✓	✓	Key-value memory	**86.1 ± 0.01**	**86.4 ± 0.02**	**85.9 ± 0.03**	**83.4 ± 0.01**

Table 5.3: Training time and memory complexity. $|Q|$ refers to the total number of questions, v is the input embedding dimension, N is the total number of latent concepts, h is the RNN hidden state dimension, S is the input sequence length, d_k is the attention key dimension, and H is the total number of attention heads.

Method	Time Complexity	Memory Complexity
DKVMN [201]	$O(2\|Q\| \times v)$	$O(2N \times v)$
DKT+Forget [121]	$O(2\|Q\| \times v)$	$O(h)$
GKT [122]	$O(\|Q\|^2 \times v)$	$O(N^2)$
SAKT [126]	$O(S^2 \times k)$	$O(H \times d_k)$
AKT [45]	$O(2S^2 \times k)$	$O(2H \times d_k)$
HawkesKT [180]	$O(2N \times v)$	$O(4N \times v)$
DGMN (ours)	$O(2\|Q\| \times v)$	$O(N^2)$

5.5.2 Ablation Study

To answer the question *Q2*, we conduct an ablation study. We compare different variants of our DGMN model to evaluate the impact of the LCG graph module, forget gating mechanism, and question ranking technique on the performance of DGMN. These variants are as follows:

- **DGMN-Basic**: This variant only includes a basic KV memory without using the LCG graph module, the forget gating mechanism, and the question ranking technique.
- **DGMN-RNN**: This variant replaces the key-value memory component in the DGMN-Basic variant with a simple RNN memory structure to store the knowledge state during answer prediction. It does not include the LCG graph module, the forget gating mechanism, nor the question ranking technique.
- **DGMN-StaticGraph**: This variant augments the DGMN-Basic variant with a static concept graph which is built upon concept-concept co-occurrence frequencies in the answer history. We follow a statistical approach similar to [122], in which node embeddings are pre-trained to predict the co-occurrence between concept pairs in the answer sequences (i.e., number of times a concept c_i appeared after a concept c_j). This variant does not include the forget gating mechanism, nor the question ranking technique.
- **DGMN-NoForget**: This variant is obtained by removing the forget gating mechanism and the question ranking technique from DGMN. The purpose of this variant is to evaluate the impact of using latent concept dependencies extracted from the LCG graph on the performance of DGMN.
- **DGMN-NoGraph**: This variant is obtained by removing the LCG graph module and the question ranking technique from DGMN. Note that, the question ranking technique has to be removed in this variant because it depends on the degree information from the LCG graph. The purpose of this variant is to evaluate the impact of the forget gating mechanism on the performance of DGMN.
- **DGMN-NoRank**: This variant includes all the components of DGMN except for the question ranking technique mentioned in Section 5.3.5. This variant is to evaluate the impact of question ranking on the performance of DGMN.

Table 5.4 summarizes the average AUC results for different variants of the DGMN model. From Table 5.4, we have the following findings:

(1) The impact of the LCG graph module can be observed by comparing DGMN-Basic and DGMN-NoForget. A statistically significant, i.e., *p-value*< 0.05, performance enhancement by a margin of 0.9%, 1.0%, 0.9%, and 0.7% is achieved for *ASSISTments2009*, *Statics2011*, *Synthetic-5* and *Kddcup2010*, respectively.

(2) The impact of the forget gating mechanism can be observed by comparing DGMN-Basic and DGMN-NoGraph. DGMN-NoGraph achieves a statistically significant performance enhancement upon DGMN-Basic by a margin of 3.3%, 2.6%, 1.8%, and 2.4% for *ASSISTments2009*, *Statics2011*, *Synthetic-5* and *Kddcup2010*, respectively.

Table 5.4: The Average AUC results for comparing different variants of DGMN over all the datasets.

Model	Component				Dataset			
	Forget	Graph	Rank	Memory	ASSISTments2009	Statics2011	Synthetic-5	Kddcup2010
DGMN	✓	Dynamic	✓	Key-Value	**86.1 ± 0.01**	**86.4 ± 0.02**	**85.9 ± 0.03**	**83.4 ± 0.01**
DGMN-NoForget	×	Dynamic	×	Key-Value	82.8 ± 0.01	83.9 ± 0.03	83.8 ± 0.04	80.7 ± 0.03
DGMN-NoGraph	✓	×	×	Key-Value	85.2 ± 0.02	85.5 ± 0.01	84.7 ± 0.02	82.4 ± 0.01
DGMN-NoRank	✓	Dynamic	×	Key-Value	85.7 ± 0.02	85.9 ± 0.01	85.1 ± 0.02	82.7 ± 0.03
DGMN-Basic	×	×	×	Key-Value	81.9 ± 0.01	82.9 ± 0.03	82.9 ± 0.01	80.0 ± 0.02
DGMN-RNN	×	×	×	RNN	72.8 ± 0.08	73.9 ± 0.07	73.6 ± 0.04	76.2 ± 0.03
DGMN-StaticGraph	×	Static	×	Key-Value	75.4 ± 0.09	76.1 ± 0.08	75.7 ± 0.05	78.3 ± 0.05

(3) The impact of the question ranking technique can be observed by comparing DGMN-NoRank with DGMN. A significant performance margin of 0.4%, 0.5%, 0.8%, and 0.7% exists between these two models for *ASSISTments2009*, *Statics2011*, *Synthetic-5*, and *Kddcup2010*, respectively.

(4) The impact of using a simple RNN memory structure on the performance can be seen by comparing DGMN-RNN against DGMN-Basic which uses attention memory. Significant performance margins of 9.1%, 9.0%, 9.3%, and 3.8% are obtained by DGMN-Basic for datasets *ASSISTments2009*, *Statics2011*, *Synthetic-5*, and *Kddcup2010*, respectively. Using a simple RNN memory structure limits the model's ability to capture the knowledge state dynamics across multiple learning concepts, whereas this can be supported by attention memory.

(5) The impact of using a static concept graph can be observed by comparing DGMN-StaticGraph and DGMN-NoForget. Significant performance margins of 7.4%, 7.8%, 8.1%, and 2.4% are gained by DGMN-NoForget using a dynamic concept graph for *ASSISTments2009*, *Statics2011*, *Synthetic-5*, and *Kddcup2010*, respectively. Although a static concept graph may also capture the relationships between learning concepts, it lacks adaption to changes according to a student's knowledge state over time, which is an advantage of the dynamic variant.

5.5.3 Analysis of LCG Graphs

To answer the question *Q3*, we visualize the LCG graphs learned from two datasets – *ASSISTments2009* and *Statics2011*. We label the nodes with their corresponding latent concept numbers, and the edges between nodes reflect relationships. In addition to this, we cluster the nodes that are close to each other in a node embedding space. Different clusters are highlighted using different colors. We analyze how these clusters reflect semantic relatedness between latent concepts.

We visualize the LCG graphs learned by DGMN from two of the datasets: *ASSISTments2009* and *Statics2011* in Figure 5.4. We apply the following principles to construct a LCG graph: 1) assigning a unique number for each latent concept; 2) clustering similar latent concepts based on their embeddings in the embedding space of $\mathbf{M}^k$; 3) highlighting latent concepts in the same cluster with the same color and edges with the colors of their incident nodes with higher degrees.

We notice that latent concepts that are close in the embedding space have closely related meanings in these datasets. This reflects the effectiveness of embedding representations for latent concepts in DGMN. In the LCG graph for *ASSISTments2009* in Figure 5.4.a, there are 10 clusters of latent concepts. In the blue cluster that represents latent concepts for solving a system of equations, one can see that concepts with the numbers $\{26, 65, 97\}$ are close to each other and their text descriptions are "*Equation solving two or fewer steps*", "*Scientific notation*", and "*Choose an equation from given information*", respectively. Similarly, in the LCG graph for *Statics2011* in Figure 5.4.b, there are 7 clusters of latent concepts. In the purple cluster that represents latent concepts about forces and their mechanical laws, the concepts with the numbers $\{17, 82, 35\}$ are close to each other, and they have the descriptions "*recognize concurrent forces*", "*rotation of forces*", and "*moving force perpendicular to the line of action*", respectively.

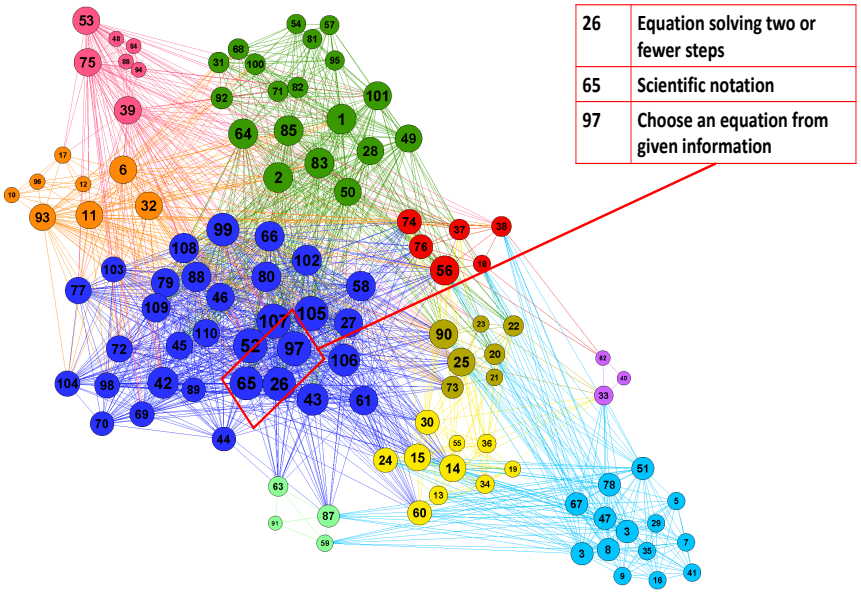
These LCG graphs also reflect the relationships between latent concepts. In the LCG graph for *ASSISTments2009* in Figure 5.4.a, the edges $\{(93, 77), (87, 61), (39, 92)\}$ represent relationships between the latent concepts ("*rotation*", "*3D figures*"), ("*algebraic solving*", "*greatest common factor*"), and ("*box and whisker*", "*range*"), respectively. It can be observed that the blue cluster that represents "*solving system of equations*" is a fundamental one for many other clusters such as "*geometric theorems*", "*statistics*", or "*probability*". Similarly, in the LCG graph for *Statics2011* in Figure 5.4.b, the edges $\{(6, 42), (12, 11), (27, 46)\}$ reflect relationships between the latent concepts ("*distinguish rotation translation*", "*rotation sense of force*"), ("*represent interaction on contacting body*", "*draw force on body*"), and ("*static problem force and moment*", "*moment sign sense relation*"), respectively. The purple cluster is densely connected to all the other clusters as it contains fundamental latent concepts such as "*forces*", while the other clusters represent operations related to it such as "*rotations*", "*moments*", or "*motion*".

5.5.4 Analysis of Forget Modelling

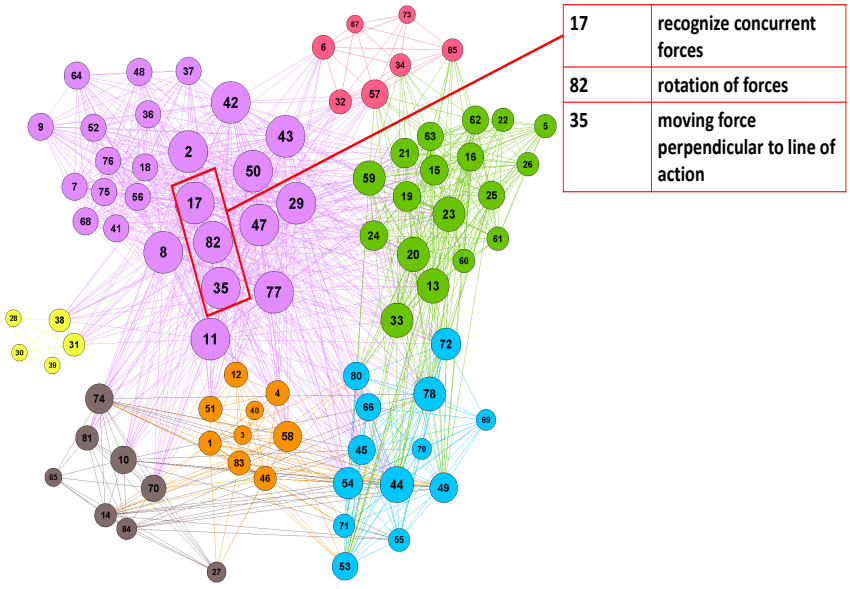
To answer the question *Q4*, this experiment aims to discuss the impact of modeling a student's forgetting behaviors over multiple latent concepts in comparison to doing it assuming only a single latent concept. To achieve this, we compare the probabilities of correctly answering questions between DGMN and DKT+forget models using a sample question sequence from the *ASSISTments2009* dataset. We visualize their differences using a heatmap (will be depicted in Figure 5.5).

To assess the effectiveness of our forgetting modeling, we examine the probabilities of predicting correct answers by the DGMN and DKT+forget models. Figure 5.5 shows a heatmap in which the question answering sequence is taken from the dataset *ASSISTments2009*. We spot two cases that highlight the differences in forgetting modeling over multiple latent concepts considering their mutual relationships and over a single latent concept. For clarity, these two cases are marked with the orange and red rectangles in Figure 5.5, respectively.

- For the first case, DKT+forget produces a lower prediction probability for q_{18} because forgetting features assuming one latent concept are *Time Lapse*=8 and *Trials*=2. In contrast, DGMN produces a higher prediction probability for q_{18} as DGMN is able to



(a)



(b)

Figure 5.4: Learned LCG graphs in two datasets: (a) *ASSISTments2009*; (b) *Statics2011*, where the sizes of nodes are proportional to their degrees and the colors reflect the clusters they belong to (best in colors).

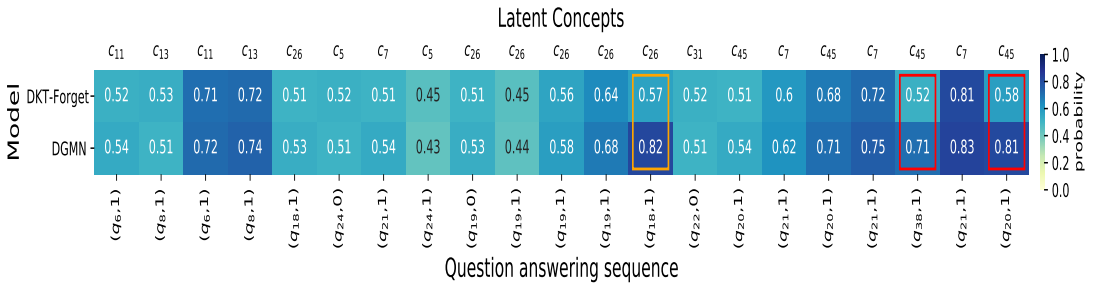


Figure 5.5: A heatmap comparing the probabilities of correctly answering questions, predicted by the DKT+forget and DGMN models, where the question answering sequence is taken from the *ASSISTments2009* dataset (best in colors).

detect that q_{19} and q_{18} relate to the same latent concept c_{26}, thereby yielding a lower forgetting chance with *Time Lapse*=1 and *Trials*=6.

- For the second case, DKT+forget predicts lower probabilities for questions q_{38} and q_{20} because the former question it was the first time to appear in the sequence, and the latter question indicates a higher forgetting chance due to *Time Lapse*=4 and *Trials=3*. In contrast, DGMN predicts higher probabilities because given that the questions q_{38} and q_{20} relate to the same concept c_{45}, this yields a lower forgetting chance with *Time Lapse*=2 and *Trials*=4.

These findings support the effectiveness of modeling forgetting features over multiple concepts while considering their relationships captured by the LCG module.

5.6 Summary

This chapter introduced a novel model that considers the effects of student forgetting behaviors during knowledge estimation. The proposed model leverages knowledge state dynamics from external memory structures to construct a graph for learning concepts and their relationships while considering forgetting behaviors. Thus, it can predict student learning performance based on both knowledge temporal dynamics and relationships across learning concepts. We compared our model with the state-of-the-art KT models on four well-established datasets. The results showed that our model outperforms all other models on all datasets. Despite the promising results achieved by the DGMN model, we highlight several limitations to be considered as follows. First, our model assumes that the total number of existent concepts is known in prior as ground truth information. While this is a common assumption in comparison to other methods (e.g., DKVMN [201], SAKT [126], and AKT [45]) it could be limiting in scenarios where such ground truth is missing. In such situations, we recommend to determine this information through an empirical analysis. Second, we note that in comparison to memory-based KT methods such as DKVMN [201], our method has an added computational cost for performing the graph convolution operation. Nevertheless, we minimized this additional cost by reusing learned embedding by the memory component to form node embedding in the LCG graph.

Third, our model is following an undirected graph approach for capturing the relationships among concepts. Although this showed the effectiveness of representing mutual concept relationships in a knowledge space, it cannot capture prerequisite dependencies among concepts that might need to consider in a directed version of the graph.

Chapter 6

Learning Data Teaching Using Knowledge Tracing

6.1 Overview

A teaching strategy can be used for teaching learning materials, exercises, and problem-solving skills to enable students to achieve their learning objectives. Typically, this is performed by observing a student's knowledge progress over essential learning concepts in a learning task, e.g., addition, subtraction, and multiplication in an elementary math course. Therefore, a human teacher can evolve such a teaching strategy according to a student's performance level, which is known as the zone of proximal development (ZPD) or "scaffolding" strategy in the student psychology studies [146, 186]. This evolution of a teaching strategy is the key to unlocking students' full potential at different levels and begs the question: *Can a machine effectively learn to teach through tracing a student's knowledge states?* To answer this question, we define an experimental setup for evaluating computational teaching models, which avoids the challenges of running repeated evaluations on human students. Thus, in this chapter, we evaluate teaching performance on computational student models instead of human ones. Specifically, we focus on machine learning student models for their needs to learn from data, similar to human students' needs to learn from learning materials. Moreover, training data teaching is an interesting problem to solve in the machine learning domain for designing effective and sample-efficient training procedures for machine learning models [203, 104, 158]. In addition, we follow a Reinforcement Learning (RL) paradigm [164] to optimize our proposed computation teaching model. Thus, in the remainder of this chapter, we refer to it as a teaching agent.

Our data teaching scenario assumes an interaction between a teaching agent and a student model working together to solve a supervised machine learning task, where the teaching agent learns to sample labeled training data for the student model from a training dataset, and the student model uses the given training data to optimize its decision function's parameters. In the past years, a number of attempts have been made to optimize the training procedure of a machine learning model. Curriculum learning methods [14, 159, 49] aimed at ranking training samples based on their difficulty levels to build a good learning curriculum for a student model. Similarly, self-paced learning (SPL) methods [87, 94, 73] used a hardness threshold that gradually increases with the progress of a student to build a training data curriculum. Machine teaching methods [104, 99] focused on selecting optimal training samples that minimize

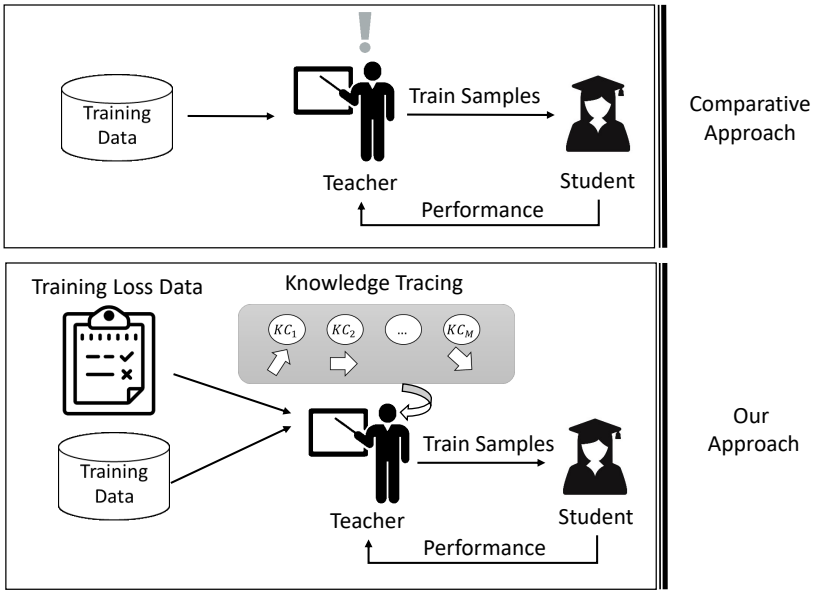


Figure 6.1: Comparing the proposed KADT method with a conventional data teaching method.

a teaching cost (e.g., the size of a training set). Despite considerable progress, these methods either depend on heuristic rules (e.g., hardness or difficulty thresholds) or assume a pre-trained student model to drive a teaching strategy.

Recently, RL has been adopted to develop teaching agents [41, 192]. It involves two building blocks: a teacher agent and a student model. A teacher agent aims to optimize a teaching strategy, while a student model follows the teaching strategy to optimize its learning objective. Despite being effective, these RL teaching methods have several limitations. First, they depend on hand-crafted states, ignoring the fact that a student model may have different performances on different learning concepts (i.e., class labels) in a supervised learning task. Second, they require a careful assignment of a target performance (e.g., classification accuracy) threshold for each learning task based on sparse reward functions that positively reward a teacher agent only if a student model performs above a specified performance threshold. This demands task-specific expertise during the RL training to land on an effective teaching policy [164]. To address these limitations, we propose a novel method for developing data teaching strategies, namely *Knowledge Augmented Data Teaching* (KADT). At its core, the KADT method has a powerful representation learning ability to capture a student model's performance by leveraging knowledge tracing techniques [137, 201, 1]. Specifically, the KADT method employs a key-value memory architecture to learn the knowledge progress of a student model in terms of learning concepts underlying a supervised learning task. This offers several learning advantages: (a) relaxing the sampling assumptions for a teaching strategy by representing a student's learning progress over learning concepts instead of individual samples; (b) capturing the temporal dynamics of a student's progress during the teaching procedure.

To highlight the difference between our proposed method and existing data teaching methods, we depict a high-level comparison in Figure 6.1. The top figure shows a commonly-used data teaching method that directly conditions its data sampling technique on the performance

of a student model, making it challenging to estimate the sampling parameters accurately. In contrast, our method at the bottom represents a supervised learning task from a knowledge tracing (KT) perspective, which assumes a set of learning concepts to impact the student's learning progress. Examples of such learning concepts could be image class labels (e.g., animals, plants, or furniture) in an image classification task or movie genres in a recommendation task. Tracing a student's learning progress provides a clear view of the areas of improvement, thereby leading to the development of a better data sampling strategy.

Furthermore, the KADT method incorporates several novel RL designs to exploit a student model's capacity and develop a data teaching strategy that matches its capacity to help it perform as best as possible. To summarize, the main contributions of this chapter are listed below:

- We propose a novel teaching method KADT which enables a coupled optimization of knowledge representation learning and teaching strategy learning through the interaction between a student model, a knowledge tracing model, and a teaching agent.
- We devise a knowledge representation learning technique that can dynamically trace the performance of a student model over learning concepts in *any* supervised learning task.
- We propose an efficient attention-pooling mechanism that distills a state representation from knowledge representations over class labels while accounting for the importance of individual training samples.
- We propose a better action representation that directly samples training data based on a student's knowledge state.
- We design a formulation for the teaching reward function that makes it dense over the state space, facilitating more training signals to the teaching agent.
- We evaluate the KADT method on four different kinds of learning tasks against the state-of-the-art methods. The results empirically validate that the KADT method consistently outperforms the other methods on all the tasks.

The remainder of this chapter is organized as follows. Section 6.2 presents the problem definition. Section 6.3 describes our methodology. Section 6.4 introduces the experimental design. Section 6.5 discusses the evaluation results. Finally, Section 6.6 concludes the chapter.

6.2 Problem Definition

A supervised machine learning problem typically involves multiple aspects, including a training set $D^{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{|D^{\text{train}}|}$ consisting of samples $\mathbf{x}_i$ from a sample space X and their labels y_i from a label space Y, a hypothesis function $h_\theta : X \to \hat{Y}$ parameterized by θ, which maps a sample space X to an estimated label space $\hat{Y}$, and a loss function L that measures the error between estimated label $\hat{y}$ and true label y. The learning objective is to find the optimal parameters of the hypothesis function h_θ that minimize a loss function over training data:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathrm{L}(h_\theta, D^{\text{train}}) \tag{6.1}$$

where Θ is a parameter search space for the hypothesis function. We call such a hypothesis function *a student model*, which can be any supervised machine learning model (e.g., a deep neural network or a simple linear regression model), and Equation 6.1 represents its optimization objective.

To teach a student model in solving a supervised learning problem (i.e., guide the search for θ^*), an effective data teaching strategy needs to be found. Such a data teaching strategy g_ω parameterized by ω, samples a sequence of training mini-batches $\mathcal{D} = \{\hat{D}^1, \hat{D}^2, \ldots, \hat{D}^{|\mathcal{D}|}\}$ from the training set D^{train} for a student model h_θ to maximize its performance on a validation set D^{valid} measured by a performance metric η. This objective can be formulated as the following optimization problem:

$$\omega^* = \arg\max_{\omega \in \Omega} \eta(h_\theta, g_\omega(D^{\text{train}}), D^{\text{valid}}) \tag{6.2}$$

where Ω is a parameter search space for teaching strategies.

In a data teaching scenario, Equation 6.1 can be re-formulated as follows to reflect training data sampled under an optimum data teaching strategy g_{ω^*}:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathrm{L}(h_\theta, g_{\omega^*}(D^{\text{train}})) \tag{6.3}$$

In this chapter, we propose a reinforcement learning method to learn such a data teaching strategy g_{ω^*} through interaction between a teacher agent and a student model.

6.3 Methodology

In this section, we present our proposed method named *Knowledge Augmented Data Teaching* (KADT) for evolving teaching strategies guided by knowledge tracing. Figure 6.2 illustrates the architecture of the KADT method. The workflow of KADT involves three components, including 1) a knowledge tracing (KT) model (top left area), 2) a student model (top right area), and 3) a teacher agent (bottom area). The student model is a supervised machine learning model, which is handled in a black box fashion (i.e., giving input and getting a loss output) without dependency on its internal design details. The KT model traces the learning progress of the student model during training and captures a state representation to reflect the student model's knowledge state. The state representation is memorized using a key-value memory, where the key matrix represents the embedding of the learning concepts in the task, and the value matrix stores the knowledge state over them (i.e., one memory slot for each learning concept). Based on the state representation provided by the KT model, the teacher agent learns a teaching policy in a reinforcement learning framework. Mini-batches of training data are sampled and guided by a reward function that encourages to have a positive change in the student's learning progress quantified by a performance metric (e.g., classification accuracy) over validation data. Below, we discuss the KADT method in detail.

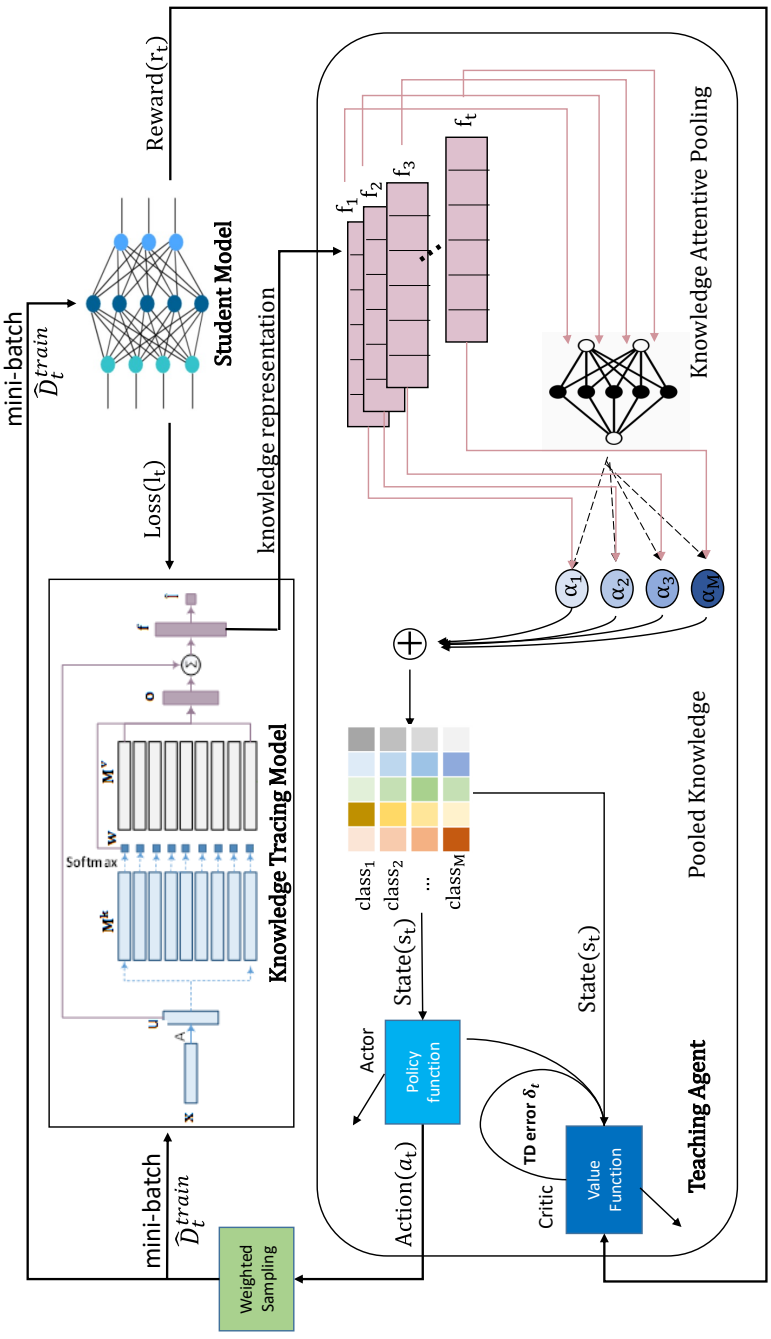


Figure 6.2: Architecture of the KADT method, which has three main components: a student model, a knowledge tracing model, and a teaching agent.

6.3.1 Student Knowledge Tracing

The performance of a student model largely depends on how it acquires knowledge from different learning concepts involved in a task, e.g., movie genres in a classification task for rating movies. Thus, inspired by knowledge tracing methods [201, 1], we design a memory-augmented knowledge tracing (KT) model to capture the knowledge representation of a student model based on its past interaction history.

Concretely, the KT model has a key-value memory structure $\mathbf{M} = \langle \mathbf{M}^k, \mathbf{M}^v \rangle$, where $\mathbf{M}^k \in \mathbb{R}^{N \times d_k}$ is a static matrix, called the *key matrix*, and $\mathbf{M}^v \in \mathbb{R}^{N \times d_v}$ is a dynamic matrix, called the *value matrix*. Let $C = \{c_1, \ldots, c_N\}$ be N learning concepts underlying the knowledge representation of a student model. Then $\mathbf{M}^k$ stores the encoding keys that represent learning concepts and $\mathbf{M}^v$ stores the performance information of a student model for each learning concept, which is dynamically changing over time. The dimension N reflects the number of learning concepts in the task, while d_k and d_v are dimensions of the memory slots. Based on our empirical analysis, we set $d_k = d_v = 50$, while N is set differently for each dataset (see further discussion in Section 6.4.1).

Given a sample $\mathbf{x}$ from the training mini-batch $\hat{D}_t^{\text{train}}$ at the time step t, we get its embedding vector $\mathbf{u} \in \mathbb{R}^{d_k}$ by embedding its one-hot encoding vector $\mathbf{ffi}(\mathbf{x}) \in \mathbb{R}^{|D^{\text{train}}|}$ with an embedding matrix $\mathbf{A} \in \mathbb{R}^{|D^{\text{train}}| \times d_k}$. Then, a dot product between $\mathbf{u}$ and each key slot $\mathbf{M}^k(i)$ in the key matrix $\mathbf{M}^k$ is supplied to a Softmax layer to get the *relevancy vector* $\mathbf{w} \in \mathbb{R}^N$:

$$\mathbf{w}(i) = \text{Softmax}(\mathbf{u}^\intercal \mathbf{M}^k(i)) \tag{6.4}$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$. Intuitively, the relevancy vector $\mathbf{w}$ reflects the relevance between the sample x and the learning concepts in $\mathbf{M}^k$.

After calculating the relevancy vector $\mathbf{w}$, the KT model proceeds in two stages. First, it reads from the value matrix $\mathbf{M}^v$ using $\mathbf{w}$ to predict the expected loss of the student model. Second, it updates $\mathbf{M}^v$ after acquiring the actual loss from the student model. We call these stages the *read stage* and the *write stage*, respectively, and discuss them further in detail.

6.3.1.1 Read Stage

In the read stage, the KT model retrieves the student's performance information with regard to the sample x to predict its expected loss. First, the relevancy vector $\mathbf{w}$ is used to calculate a weighted sum from the memory slots of the value matrix $\mathbf{M}^v$, which yields the *read vector* $\mathbf{r} \in \mathbb{R}^{d_v}$:

$$\mathbf{r} = \sum_{i=1}^{N} \mathbf{w}(i) \mathbf{M}^v(i) \tag{6.5}$$

Then, the read vector $\mathbf{r}$ is concatenated with the embedding vector $\mathbf{u}$ of the sample x, and then fed to a Tanh layer to calculate the *representation vector* $\mathbf{f} \in \mathbb{R}^N$:

$$\mathbf{f} = \text{Tanh}(\mathbf{W}_1^\intercal [\mathbf{r}, \mathbf{u}] + \mathbf{b_1}) \tag{6.6}$$

where $\text{Tanh}(z_i) = (e^{z_i} - e^{-z_i})/(e^{z_i} + e^{-z_i})$, $\mathbf{W}_1 \in \mathbb{R}^{(d_v+d_k)\times N}$ is the weight matrix of the Tanh layer, and $\mathbf{b}_1$ is a bias vector. This representation vector captures a student's current knowledge progress over sample x and the sample information itself.

Finally, we send the representation vector $\mathbf{f}$ to a linear function as input and take the dot product of the output vector with $\mathbf{ffi}(\mathbf{x})$ to predict the estimated loss $\hat{\text{l}}^{\text{student}}$ of the student model for the sample $\mathbf{x}$ as follows:

$$\hat{\text{l}}^{\text{student}} = (\mathbf{W}_2^{\intercal}\mathbf{f} + \mathbf{b}_2) \cdot \mathbf{ffi}(\mathbf{x}) \tag{6.7}$$

where $\mathbf{W}_2 \in \mathbb{R}^{N\times|D^{train}|}$ is a weight matrix, $\mathbf{b}_2$ is a bias vector, and $\hat{\text{l}}^{\text{student}}$ is a scalar value.

6.3.1.2 Write Stage

After the student model predicts the class label $\hat{y}$ of $\mathbf{x}$, we acquire the actual loss $\text{l}^{\text{student}} = \ell(\hat{y}, y)$, where $\ell(\cdot)$ is the loss function of the student model. Then, the value memory $\mathbf{M}^v$ is updated to reflect the student model's current performance via the write stage.

Specifically, for each sample $\mathbf{x}$ and its prediction validity status $\epsilon \in \{0, 1\}$, we embed them using an embedding matrix $\mathbf{B} \in \mathbb{R}^{(2|D^{\text{train}}|\times d_v)}$ to get an embedding vector $\mathbf{j} \in \mathbb{R}^{d_v}$. Afterward, we concatenate $\mathbf{j}$ with the representation vector $\mathbf{f}$ to add the corresponding performance information. The resulting write vector $\mathbf{v} = [\mathbf{f}, \mathbf{j}] \in \mathbb{R}^{N+d_v}$ is used to update the value memory $\mathbf{M}^v$ through erase and add signals [52].

Based on the write vector $\mathbf{v}$, we update the value matrix $\mathbf{M}^v$. This is done through erasing the existing information from $\mathbf{M}^v$ using an erase signal $\mathbf{e} \in \mathbb{R}^{d_v}$, and then adding new information to $\mathbf{M}^v$ using an addition signal $\mathbf{a} \in \mathbb{R}^{d_v}$.

Let $\mathbf{W}_e \in \mathbb{R}^{(N+d_v)\times d_v}$ be a weight matrix and $\mathbf{b}_e$ is a bias vector. The *erase signal* is calculated as follows:

$$\mathbf{e} = \text{Sigmoid}(\mathbf{W}_e^{\intercal} \cdot \mathbf{v} + \mathbf{b}_e) \tag{6.8}$$

where $\text{Sigmoid}(z_i) = 1/(1 + e^{-z_i})$. Let $\mathbf{1}$ be a row vector of all ones, and the updated value matrix $\tilde{\mathbf{M}}^v_{\text{updated}}$ is calculated using element-wise multiplication as follows:

$$\tilde{\mathbf{M}}^v_{\text{updated}}(i) = \mathbf{M}^v(i)[\mathbf{1} - \mathbf{w}(i)\mathbf{e}] \tag{6.9}$$

Thus, the i_{th} slot of $\mathbf{M}^v$ is erased (i.e., set to zero) if the corresponding values in the relevancy vector $\mathbf{w}$ and the erase signal $\mathbf{e}$ are both equal to one and remains unchanged if either of them is zero.

Let $\mathbf{W}_a \in \mathbb{R}^{(N+d_v)\times d_v}$ be a weight matrix. The *addition signal* $\mathbf{a}$ is calculated as below:

$$\mathbf{a} = \text{Tanh}(\mathbf{W}_a^{\intercal} \cdot \mathbf{v} + \mathbf{b}_\mathbf{a}) \tag{6.10}$$

Finally, the value matrix $\mathbf{M}^v$ is updated as:

$$\mathbf{M}^v(i) = \tilde{\mathbf{M}}^v_{\text{updated}}(i) + \mathbf{w}(i)\mathbf{a} \tag{6.11}$$

6.3.1.3 Model Optimization

To optimize the KT model's parameters (i.e., the embedding matrices, weight, and bias parameters of different neural layers), we utilize the *Root Mean Square Error* (RMSE) function to calculate the differences between the estimated loss $\hat{\mathrm{l}}^{\mathrm{student}}$, and the actual loss $\mathrm{l}^{\mathrm{student}}$, acquired after the student model predicts the class label for each sample in $\hat{D}_t^{\mathrm{train}}$.

The KT model is trained using the following loss function:

$$\mathcal{L}_{\mathrm{KT}} = \sqrt{\frac{\sum_{i=1}^{|\hat{D}^{\mathrm{train}}|}\left(\hat{l}_i^{student} - l_i^{student}\right)^2}{|\hat{D}^{\mathrm{train}}|}} \tag{6.12}$$

After calculating the RMSE error based on the current training mini-batch, the parameters of the KT model are updated using gradient descent through back-propagation.

6.3.2 Data Teaching Strategies

We design a teacher model in a reinforcement learning framework called *teaching agent*. The teaching agent aims to optimize a *teaching policy* (i.e., a data teaching strategy) for a student model guided by the knowledge representation of a student model learned by the KT model.

6.3.2.1 Teaching Interactions

Following the reinforcement learning paradigm [10, 11], we model teaching interactions as a Markov decision process (MDP), represented by a tuple (S, A, T, R). Here, S is a state space, A is an action space, $T : S \times A \times S \rightarrow [0, 1]$ is a state transition function such that $T(s_t, a_t, s_{t+1}) = P(s_{t+1}|a_t, s_t)$ represents transition probabilities between states after executing actions, and $R : S \times A \rightarrow \mathbb{R}$ is a reward function. Below, we discuss them in detail.

States. Each state $s_t \in S$ in our work is represented as a matrix $s_t \in \mathbb{R}^{O \times N}$, where O is the number of class labels, and N is the number of learning concepts in a learning task. Intuitively, each row in this matrix represents the knowledge of a student model corresponding to a class label, which is pooled from the representation vectors of its training samples w.r.t. the underlying learning concepts (i.e., columns).

Specifically, we distill the latest knowledge representation of a given class label y by calculating a *pooled knowledge vector* $g_t^y \in \mathbb{R}^N$ from its corresponding elements in the set of knowledge vectors F_t of training samples in the lately sampled mini-batch $\hat{D}_{t-1}^{train}$. The set of knowledge vectors F_t is projected by applying the read stage of the KT model on mini-batch $\hat{D}_{t-1}^{train}$ (see Section 6.3.1). In the first teaching interaction, the mini-batch $\hat{D}_{t-1}^{train}$ is sampled using a uniform random distribution across all class labels and training samples (i.e., sampling from all class labels with an equal probability and treating samples in each class label as being equally likely); afterward, it is the lately sampled mini-batch by our teaching agent. We follow an attentive pooling method to calculate g_t^y as follows:

$$s_t(y) = g_t^y = EMA(\sum_{i=1}^{I} \alpha_t^i \mathbf{f}_t^i) \quad \forall y \in Y \tag{6.13}$$

where $EMA(u) = c\,u + (1-c)\,EMA_{t-1}$, for $c = \frac{2}{t+1}$ is an exponential moving average, I is the total number of training samples belonging to class label y in the mini-batch $\hat{D}_{t-1}^{train}$, $\alpha_t^i \in [0,1]$ is the attention weight for a sample $\mathbf{x}_i$ at time point t, $\mathbf{f}_t^i$ is the knowledge representation vector of a sample $\mathbf{x}_i$ at time point t calculated as per Equation 6.6, and Y is the set of class labels in the task.

The attention weight α_t^i for a sample $\mathbf{x}_i$ is calculated using a gated attention mechanism [7, 71], which controls the amount of information to pass from each sample to the pooled knowledge vector of its corresponding class label:

$$GA(\mathbf{f}_t^i) = \exp(\mathbf{c}^\intercal \frac{(Tanh(\mathbf{W}^\intercal \mathbf{f}_t^i) \odot Sigmoid(\mathbf{U}^\intercal \mathbf{f}_t^i)}{\sqrt{N}}) \tag{6.14}$$

$$\alpha_t^i = \frac{GA(\mathbf{f}_t^i)}{\sum_{j=1}^{I} GA(\mathbf{f}_t^j)} \tag{6.15}$$

where $\mathbf{c} \in \mathbb{R}^L$ is a learnable weight vector, $\mathbf{W} \in \mathbb{R}^{N \times L}$ is a learnable weight matrix for the $Tanh$ function, $\mathbf{U} \in \mathbb{R}^{N \times L}$ is a learnable weight matrix for the $Sigmoid$ function, operator $\odot$ is an element-wise product, and $\sqrt{N}$ is a scaling factor to control the output range and prevent gradient vanishing [177]. The $Sigmoid$ function acts as a gating filter to control the information to pass from the $Tanh$ function.

Actions. Given a state, the teaching agent selects a mini-batch of training samples and sends this mini-batch to the student model to train on during each interaction. An action $a_t \in \mathbb{R}^O$ is a *sampling vector* normalized using a $Softmax$ output layer in the actor-network of the teaching agent as follows:

$$a_t = \psi(s_t \,|\, \theta^\psi) \text{ subject to } \sum_{i=1}^{O} a_t(i) = 1. \tag{6.16}$$

where $\psi(s_t \,|\, \theta^\psi)$ is the actor-network with parameters θ^ψ. Each $a(i)$ corresponds to class label y^i, and the value of $a(i)$ indicates the percentage of the next mini-batch size dedicated to samples from the class label y^i.

Weighted Sampling. To sample the next mini-batch, we utilize a *weighted sampling* method that takes action a_t and the latest attention weights (Equation 6.15) over training samples in each class label and outputs the mini-batch $\hat{D}_t^{train}$. Instead of uniformly sampling with percentage $a_t(i)$ from class label y^i, we propose a sampling approach that counts for essential samples within class label y^i that could advance a student's learning progress over the coming teaching interactions.

Our sampling approach works efficiently by distilling sample weights of a given class label only using its training samples presented in the recent mini-batch. We employ an estimation technique to get the weights of the remaining samples that were not lately sampled. Our technique estimates the weights of other samples in class label y through a moving average given the weights of the present samples as follows:

$$\hat{\alpha}^u = \sum_{i=1}^{I} (x^u \cdot x^i) \times \alpha_t^i \quad \text{for } x^u \notin \hat{D}_{t-1}^{train} \tag{6.17}$$

$$\alpha_t^u = \frac{\alpha_{t-1}^u + \hat{\alpha}^u}{\Gamma} \tag{6.18}$$

where $\hat{\alpha}^u$ is a weighted sum estimate of the weight for a sample x^u in class label y, I is the total number of training samples belonging to class label y in the mini-batch $\hat{D}_{t-1}^{train}$, and Γ is the number of previous teaching interactions.

Reward Function. Let $p_t = \eta(h_{\theta^*}, \hat{D}_t^{train}, D^{\text{valid}})$ denotes the performance of a student model h_{θ^*} at the time step t after being trained on the mini-batch $\hat{D}_t^{train}$, and validated on validation set D^{valid}. Then, our reward function R is calculated to reflect the magnitude of the performance change for a student model defined as the below piecewise function :

$$R(s_t, a_t) = \begin{cases} 0 & \text{if } |p_t - p_{t-1}| \leq \epsilon; \\ (p_t - p_{t-1}) & \text{otherwise.} \end{cases} \tag{6.19}$$

In order to prevent oscillations over small variations, this reward function is designed to be ϵ-insensitive to the magnitude of the performance change, where $\epsilon \leq 0.1$ is a hyper-parameter.

The key idea behind this reward function is to positively reward actions that can improve the learning progress of a student model and penalize those that slow down the progress. The effectiveness of using the learning progress as a reward signal has been previously studied in the RL literature [125, 150]. In our work, enhancing the learning progress needs *balanced mini-batches* that contain new or challenging samples in addition to known ones to prevent forgetting. For example, selecting training mini-batches that have been correctly classified by a student model will not enhance the performance; hence, our reward function will generate a low reward value, i.e., zero, since it will lead to the same performance. On the other hand, selecting only hard training mini-batches will also decrease the performance of a student model in the long run and our reward function yields low reward values accordingly. By setting the reward function around the learning progress, we do not need to use heuristic rules (e.g., hardness thresholds) that require domain knowledge and might bias the learning process toward specific samples.

Remark. There are two possible ways for designing our reward function: sparse reward function and dense reward function. As used in L2T [41], a sparse function requires specifying a performance threshold. Only values that exceed the performance threshold yield positive reward signals; otherwise, the reward function yields zero. However, setting a performance threshold is a difficult task in real-world applications, and an unrealistic performance threshold would negatively affect the effectiveness of a teaching policy. Moreover, learning with sparse reward signals often leads to unstable behaviors and slower convergence, in comparison with using a dense reward function [164]. Therefore, we design a dense reward function that maps the performance change of a student model into a reward value rather than using a sparse reward function with a pre-defined performance threshold.

6.3.2.2 Actor-Critic Algorithm

We design our RL algorithm based on deep deterministic policy gradient (DDPG) [96], consisting of two building blocks: (1) *a critic network* $Q(s,a\,|\,\theta^Q)$ estimates the Q-value of the current state-action pairs; (2) *an actor network* $\psi(s\,|\,\theta^\psi)$ that learns to select an optimal action for the current state. We follow an actor-critic design for its reported advantage of combining sample efficiency of value-based RL algorithms (i.e., the critic part) and stability (i.e., smooth changes of action probabilities) of policy-based RL algorithms (i.e., the actor part) [196]. These properties are vital in our data teaching scenario as we need to reduce the number of teaching iterations needed to converge on a good teaching strategy while keeping the teaching process stable for a smooth student training.

Following DDPG, we use the replay buffer (RB) technique to build a mini-batch V of (state, action, reward) transitions which updates the parameters of the critic and actor networks through gradient descent. We also follow the concept of target networks which are versions of the critic and actor networks with parameter values $(\theta^{Q'}, \theta^{\psi'})$ being updated proportionally to the latest actor and critic values (θ^Q, θ^ψ) with a delay factor $\tau \ll 1$ as shown in Equations 6.20 and 6.21:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{6.20}$$

$$\theta^{\psi'} \leftarrow \tau\theta^\psi + (1-\tau)\theta^{\psi'} \tag{6.21}$$

The purpose of these target networks is to make the optimization of the actor and critic networks stable as we calculate the gradient updates based on the Q-value estimate from the critic network itself [96].

The critic network's parameters are optimized to minimize the temporal difference (TD) loss [164] shown in Equation 6.22. The actor network's parameters and the attention learnable parameters are updated through the policy gradient function [96] depicted in Equation 6.23.

$$L_{TD} = \frac{1}{|V|}\sum_{i=1}^{|V|}(Q(s_i,a_i|\theta^Q) - (r_{t+1} + \gamma \max_{a'} Q\,(s_{t+1},a'|\theta^{Q'}))^2 \tag{6.22}$$

$$\nabla_{\theta^\psi} J \approx \frac{1}{|V|}\sum_{i=1}^{|V|} \nabla_a Q(s,a\,|\theta^Q)\,|_{s=s_i,a=\psi(s_i)} \nabla_{\theta^\psi}\psi(s\,|\theta^\psi)\,|_{s_i} \tag{6.23}$$

where γ is the discounting factor and $|V|$ is the length of a transitions mini-batch sampled from the reply buffer.

The action exploration is performed randomly using the *Ornstein–Uhlenbeck* (OU) stochastic process [96], which generates temporally correlated exploration noise for a smooth transition across action values. The OU process is calculated as per Equation 6.24:

$$da_t = \theta(\beta - a_t)dt + \sigma dW_t \tag{6.24}$$

where θ, σ, and β are parameters. W_t represents the Wiener process [96], which is a stochastic process being initialized as $W_0 = 0$ and incremented by a Gaussian random value $(W_{t_i} - W_{t_j}) \sim \mathcal{N}(0, t_i - t_j)\;\forall\, 0 \le t_j < t_i$ at each time step. Algorithm 2 describes the main steps for training

our KADT method.

Algorithm 2 Training KADT

Initialize critic $Q(s,a\,|\,\theta^Q)$ with θ^Q at random
Initialize actor $\psi(s\,|\,\theta^\psi)$ with θ^ψ at random
Initialize target networks Q' and ψ' with $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\psi'} \leftarrow \theta^\psi$
Initialize KT parameters at random
Initialize reply buffer RB
for episode = 1, M **do**
    Initialize OU process da_t for action exploration
    Let $F_t \leftarrow KT.read(\hat{D}^{train}_{t-1})$ ▷ Knowledge tracing
    Let $s_t \leftarrow concat(g^y_t)\forall y \in Y$ ▷ Attentive pooling
    for $t = 1, T$ **do**
        Select $a_t = \psi(s_t\,|\,\theta^\psi) + da_t$
        Estimate $\mathbf{ff}_t$ using Eq. 6.18
        $\hat{D}^{train}_t \leftarrow WeightedSampling(a_t, \mathbf{ff}_t, D^{train})$
        Train the student on $\hat{D}^{train}_t$
        Observe reward r_t and new state s_{t+1}
        Update the student's parameters using SGD
        Update the KT parameters using Eq. 6.12
        Save transition (s_t, a_t, r_t, s_{t+1}) into RB
        Sample a random batch of V transitions from RB
        Update critic using L_{TD} in Eq. 6.22
        Update actor using policy gradient in Eq. 6.23
        Update target networks using Eq. 6.20-Eq. 6.21
    end for
    endfor
end for
endfor

3) Model Optimization:

The optimization objective of the teaching agent is to find an optimal teaching policy $\pi^* : S \rightarrow A$ that maximizes the average reward gain:

$$\pi^* = \arg\max_{\pi \subset \Pi} \frac{1}{E} \sum_{i=1}^{E} r_i^\pi \tag{6.25}$$

where Π is a teaching policy search space and E is the total number of teaching episodes. We optimize the actor-critic parameters (θ^Q, θ^ψ) using Equations 6.22 and 6.23 to find the optimal teaching policy π^*.

6.4 Experiments

We conduct experiments to evaluate the KADT method against the state-of-the-art methods, aiming to answer the following questions:

Q1: How effectively can our KADT method teach a student model across different learning tasks?

Q2: How well can our KADT method be generalized to teach different student models, i.e., the generalization ability of our teaching agent?

Q3: How well can our KADT method learn knowledge representation of a student model to improve performance?

Q4: How well can our KADT method perform in comparison to the state-of-the-art RL teaching method?

Q5: How does each main component of our KADT method affect its performance?

Below, we introduce our experimental setup for answering the above questions.

6.4.1 Datasets

We consider four different kinds of learning tasks in our experiments: a knowledge tracing task, a sentiment analysis task, a movie recommendation task, and an image classification task. Each task has a dedicated dataset (see Table 6.1). We follow a heuristics approach that assigns the number of clusters in the input feature space as a good value for the number of learning concepts (N) in each task. This approach assumes that in a supervised machine learning task, monitoring performance across all possible clusters of a feature space shall lead to better performance results. For example, in an image classification dataset, these clusters could be super-classes such as animals, humans, or plants, while in a movie recommendation task, they could be movie genres. We detail this heuristic in the description of each dataset as follows:

Knowledge Tracing on ASSISTments2009 [1] The dataset ASSISTments2009 was collected during the school year $2009-2010$ using the ASSISTments online education website [2]. It consists of 110 distinct questions answered by $4,151$ students which leads to a total number of $325,637$ exercises. The questions are represented as one-hot encoding vectors (i.e., one value in the corresponding index of each question in the dataset and zeros elsewhere). Based on previous studies conducted on this dataset [1, 201], there are 10 different learning concepts in the questions. There are two class labels in this dataset including *correct* and *incorrect*.

Sentiment Analysis on IMDB [3]**:** The IMDB [110] dataset includes movie reviews in text with a total of $50,000$ reviews. There are two class labels: *positive review* and *negative review*. The overall distribution of labels is balanced, with $25k$ for the positive review and $25k$ for the negative review. The text reviews are embedded using the word2vec embedding with a dimension size of 256. There are 11 learning concepts representing the general categories of movies reviewed in the text.

[1] https://sites.google.com/site/assistmentsdata/home/2009-2010-assistment-data
[2] https://new.assistments.org/
[3] IMDB: http://ai.stanford.edu/~amaas/data/sentiment/

Table 6.1: The setup of four learning tasks, including the corresponding student models and datasets. The sizes of mini-batches are set based on empirical analysis of validation sets.

Task	Dataset	Size	#Classes	Student Model Setup [training→Testing]		Mini-batch Size
				Similar Student Mode	Different Student Mode	
Knowledge Tracing	ASSISTments2009	325,637	2	[SKVMN→SKVMN]	[SKVMN→DKT]	256
Sentiment Analysis	IMDB	50,000	2	[LSTM→LSTM]	[LSTM→SVM]	16
Movie Recommendations	MovieLens	1000,000	2	[LSTM→LSTM]	[LSTM→MF]	1000
Image Recognition	CIFAR-100	60,000	100	[MLP→MLP]	[MLP→CNN]	128

Movie Recommendations on MovieLens [1]**:** The MovieLens dataset includes 1 million movie ratings performed by 6000 users on 4000 movies released in 2003. Each move rating has three features: user id, movie id, and timestamp. There are five class labels representing a five-star rating (i.e., from 1 to 5) and 18 learning concepts representing the movie genres. We follow the setting of Click-Through Rate (CTR) (i.e., binary labels: click and no click) by regarding labels 4 and 5 as click and labels 1 to 3 as no-click. This allows us to standardize this evaluation into a classification task.

CIFAR-100 Image Classification [2]**:** The CIFAR-100 dataset was collected by [84]. The dataset consists of 60000 RGB images of $32x32$ pixels. There are 100 class labels with 600 images for each class. These 100 classes in CIFAR-100 are grouped into 20 super-classes, and each super-class corresponds to a learning concept. Thus, we use 20 learning concepts in this dataset.

6.4.2 Baseline Methods

We discuss the baseline methods used in the teacher model setup and the student model setup in our experiments.

Teacher Model Setup. We compare the performance of the proposed method KADT against two state-of-the-art methods:

- **Self-Paced Learning (SPL) [87]:** This method works by filtering out training examples that have a loss value exceeding a predefined hardness threshold. This threshold value increases gradually during the training time.

- **Learning to Teach (L2T) [41]:** This method uses a reinforcement learning agent to sample training mini-batches for a student model. The agent optimizes a sparse reward function that yields a positive reward only when the student model's performance exceeds a predefined performance threshold.

In addition, we use random sampling as a baseline model and refer to it as "RandomTeach"

- **Random Sampling (RandomTeach):** This method provides the conventional data teaching strategy, i.e., sampling mini-batches from a uniform random distribution.

[1] MovieLens: https://grouplens.org/datasets/movielens/
[2] CIFAR-100:https://www.cs.toronto.edu/~kriz/cifar.html

For the SPL and L2T methods, we follow the same parameter configuration as described in [87] and [41], respectively. Each method has an additional hyper-parameter, i.e., the hardness threshold in SPL and the performance threshold in L2T. For SPL, we use a range of values $\{80, 100, 120, 150, 180\}$ for the hardness threshold. For L2T, we follow the heuristic strategy in [41] that sets the best performance value from the past episodes as the threshold for the following ones. We report their best results.

6.4.3 Metrics

We use two evaluation metrics to report the performance: (1) Average classification accuracy, and (2) Average area under curve (AUC) for the receiver operating characteristic (ROC) curve, which represents the ratio between the true positive (TP) rate and the false positive (FP) rate on different test cutoffs.

6.4.4 Experimental Setup

In this section, we outline the setup for our experimental evaluations as follows.

6.4.4.1 Student Model Setup

We consider two different student models for each dataset. For the knowledge tracing task, we use deep knowledge tracing (DKT) [137] and sequential key-value knowledge tracing (SKVMN) [1]. For the sentiment analysis task, we use a support vector machine with a radial basis kernel (SVM) and a long short-term memory (LSTM) [60] model. For the recommendation task, we use a matrix factorization (MF) model [143] and the same LSTM model from the sentiment analysis task. For the image classification task, we use a multi-layer perceptron (MLP) and a convolutional neural network (CNN) architecture (ResNet50) [63]. We follow the same design and hyper-parameter configuration as per the cited work for these models.

6.4.4.2 Training and Testing Strategies

We divide each dataset into a train-validate set and a test set using a $(70\% - 30\%)$ ratio. Then, the train-validate set is further divided into $\mathrm{D}^{\mathrm{train}}$ and $\mathrm{D}^{\mathrm{valid}}$ using a 5-fold cross validation.

Training process: the training process consists of two phases in our experiments:

- *Phase 1 – Teacher Training.* The purpose of this phase is to train a teacher model for learning a data teaching strategy. This is achieved by assigning a fixed number of *350* training episodes selected through empirical analysis. Each training episode has 50 time steps (i.e., episode time horizon) for the L2T and KADT methods and an equivalent number of training epochs for the SPL model. For example, in the IMDB dataset, we have a total of 50,000 samples, and then the size of the train-validate set is 35,000. Following 5-fold cross-validation, the number of training samples is 28,000, which leads to a mini-batch size of 16. The equivalent number of epochs for 350 episodes is 10. RandomTeach does not have this phase because there are no parameters to optimize for its data teaching strategy. For the reward calculation in the L2T and KADT methods,

we use 5% of the validation data. To stabilize the teacher training of the KADT model, we perform a warming-up for the KT model before the first episode that uses one epoch over the training dataset while fixing the parameters of the student model (i.e., running it in inference mode).

- *Phase 2 – Student Training.* The purpose of this phase is to apply the data teaching strategy learned in Phase 1 to a new student model. There are two different modes: (a) using a student model that is the same as the student model in Phase 1 but with re-initialized parameters (see teaching with similar students mode in Section 6.4.4.3), and (b) using a student model that is different from the student model in Phase 1 (see teaching with different students mode in Section 6.4.4.3). In this phase, the teacher model is not allowed to update its parameters (i.e., fix the teaching strategy). We give the same number of episodes and epochs as per Phase 1 to apply the data teaching strategies to the student model.

Testing process: in the testing process, we evaluate the performance of a teacher model by applying the student model trained by its data teaching strategy in Phase 2 on the test set D^{test}.

6.4.4.3 Student Evaluation Modes

To answer Q1 and Q2, we design our experiments in two modes: *teaching with similar student mode* and *teaching with different student mode*. We present the details of these two modes below:

Teaching with similar students mode: in this mode, we use the same type of student models (e.g., SVM) across Phases 1 and 2. To evaluate how effectively a data teaching strategy is learned from the training phase, we randomly initialize the parameters of a student model in the testing phase. This mode evaluates how well the teacher model can teach on new training samples (i.e., another part of the training dataset) using the same student architecture.

Teaching with different students mode: in this mode, we change the types of student models across Phases 1 and 2. For example, if we train a feed-forward neural network student model in phase 1, we replace it with a recurrent neural network student model in phase 2. This mode aims to evaluate the generalization of data teaching strategies optimized by a teacher model over different student models and training samples.

6.5 Results and Discussion

In this section, we present the results of our experiments and discuss our observations. We answer the questions Q1−Q5 in Section 6.5.1−Section 6.5.5, respectively.

6.5.1 Teaching with Similar Student Models

Table 6.2 shows the average classification accuracy achieved by the student models in the teaching with similar student mode. It can be observed that our KADT method outperforms

Table 6.2: Accuracy results for teaching with similar and different student modes averaged over five independent runs.

Dataset	Similar Students Mode				Different Students Mode			
	RandomTeach	**SPL**	**L2T**	**KADT**	**RandomTeach**	**SPL**	**L2T**	**KADT**
ASSISTments2009	81.63 ± 0.4	83.45 ± 0.02	84.31 ± 0.03	**87.10 ± 0.04**	79.82 ± 0.3	80.41 ± 0.06	81.29 ± 0.04	**84.26 ± 0.04**
IMDB	88.54 ± 0.8	88.80 ± 0.05	89.46 ± 0.06	**92.24 ± 0.05**	78.87 ± 0.5	81.04 ± 0.03	83.22 ± 0.05	**86.16 ± 0.03**
MovieLens	75.12 ± 0.6	76.45 ± 0.06	77.23 ± 0.03	**80.31 ± 0.02**	74.82 ± 0.8	76.87 ± 0.04	77.73 ± 0.06	**80.47 ± 0.03**
CIFAR-100	62.18 ± 0.4	64.68 ± 0.04	65.33 ± 0.06	**68.75 ± 0.03**	68.71 ± 0.5	70.27 ± 0.05	71.82 ± 0.03	**74.58 ± 0.02**

the other methods on all datasets with a statistically significant margin (p-value < 0.05), and it is followed by the L2T method. This shows the effectiveness of using reinforcement learning to learn a data teaching strategy in comparison to the other methods. Further, the models with learnable teaching strategies (i.e., KADT, L2T, and SPL) perform better than RandomTeach which uses the random teaching strategy. The performance of KADT gained above L2T demonstrates the effectiveness of our KT model in learning knowledge representations, in comparison to using hand-crafted states in L2T. Overall, KADT outperforms the second best-performed method L2T by a margin of 2.79, 2.78, 3.08, and 3.42 on the datasets ASSISTments2009, IMDB, MovieLens, and CIFAR-100, respectively. The performance margin on CIFAR-100 was the smallest across all datasets because CIFAR-100 is the most challenging dataset with 100 different classes and the other datasets have binary classes.

Figure 6.3 shows the AUC results, which confirm the above findings. Our KADT method achieves the highest AUC value across the four supervised learning tasks.

6.5.2 Teaching with Different Student Models

Table 6.2 also shows the average classification accuracy for the student models in teaching with different student mode. Similarly, our KADT method outperforms the L2T model (the next best performer) by a margin of 2.97, 2.94, 2.74, and 2.76. For the ASSISTments2009 and IMDB results, a noticeable performance degradation occurs due to transforming from student models (i.e., SKVMN and LSTM, respectively) to less capable models (i.e., DKT and SVM, respectively). For the IMDB dataset, the results are comparable to the previous experiment as the learning capacity of the two student models (i.e., LSTM and MF) is similar in the recommendation task. A significant enhancement can be observed in the results of the CIFAR-100 dataset in comparison to the previous experiment. This is because the image representation capacity of the CNN student model used in this experiment is better than the one using the MLP model explored previously. Despite these changes in the performance results, it can still be observed that the learnable teaching methods (i.e., KADT, L2T, and SPL) enhance the performance in comparison to the RandomTeach method. Figure 6.4 confirms the performance gain achieved by the KADT method that outperforms the next best performer L2T on the AUC metric by margins of 2.97, 2.94, 2.74, and 2.76 for the datasets ASSISTments2009, IMDB, MovieLens, and CIFAR-100, respectively.

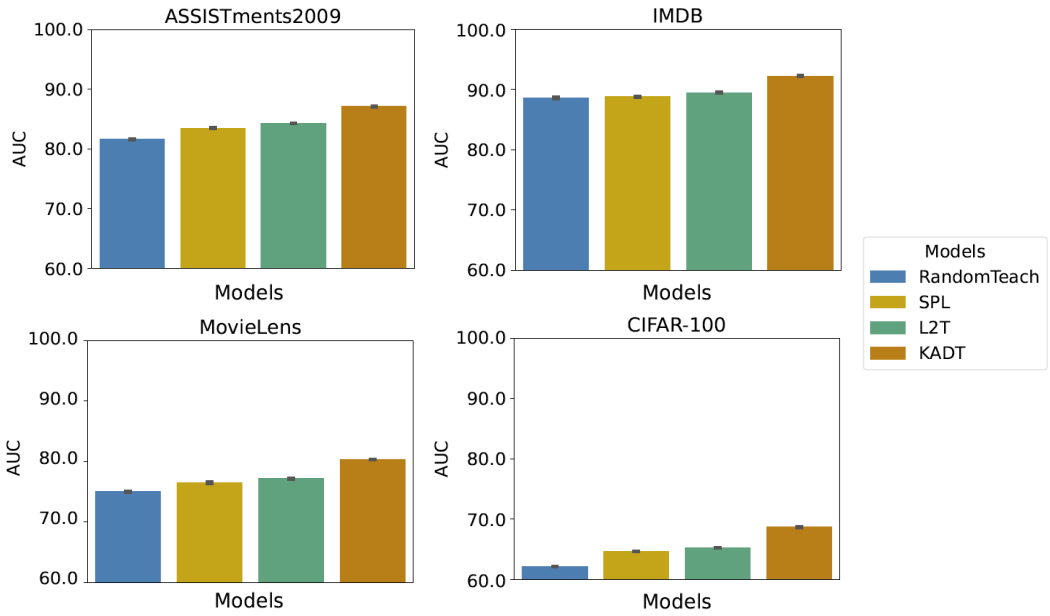


Figure 6.3: AUC results averaged over five runs for the teaching with same students mode over four datasets.

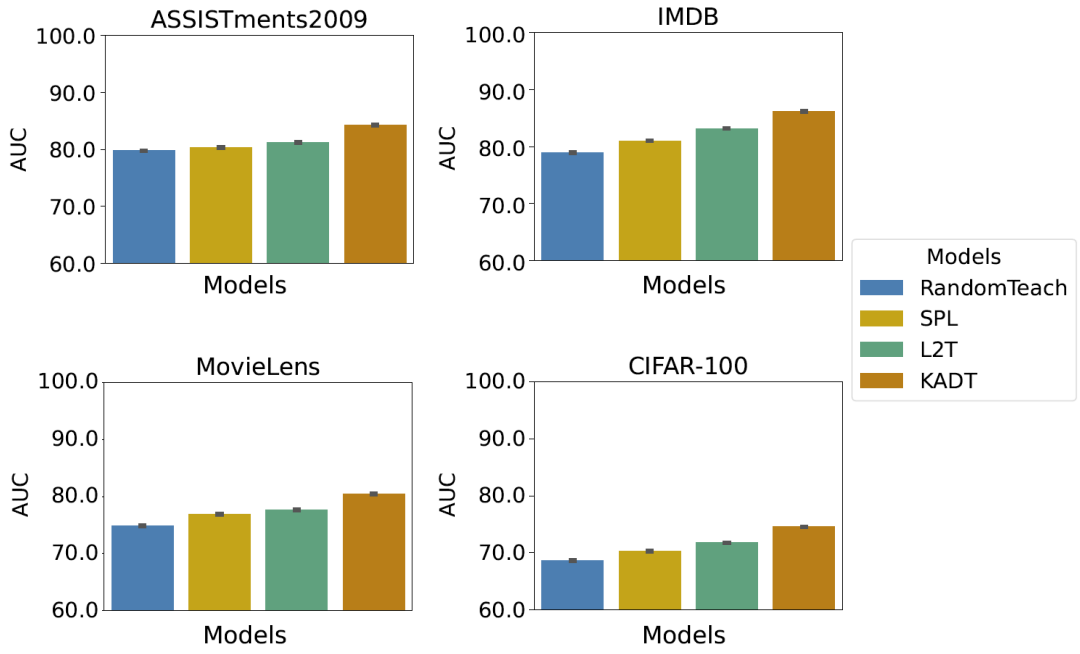


Figure 6.4: AUC results averaged over five runs for the teaching with different students mode over four datasets.

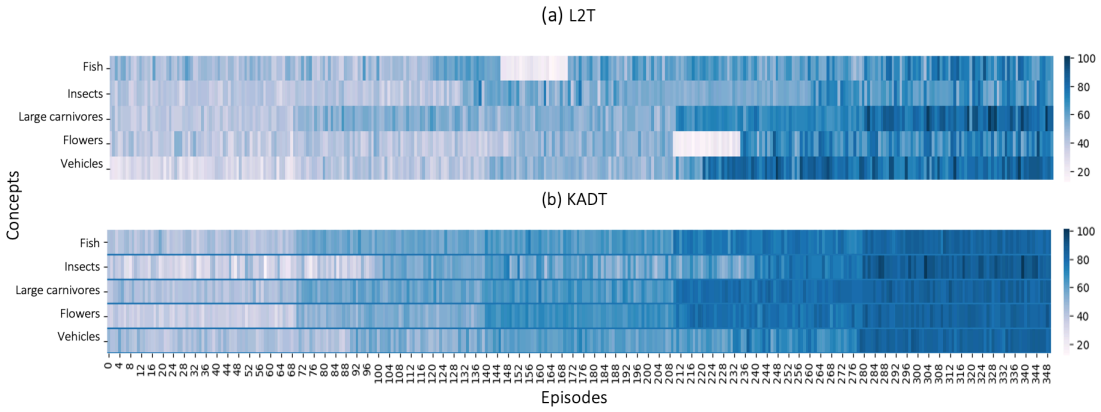


Figure 6.5: Heatmaps for the validation accuracy for a CNN student model on the CIFAR-100 dataset over five learning concepts. (a) L2T results. (b) KADT results.

6.5.3 Evaluating Knowledge Representation Learning

To evaluate the impact of a student's knowledge representation learning on the effectiveness of a data teaching strategy, we present the progress of a student model's validation accuracy by our KADT method and compare it with the L2T method in Figure 6.5. These heatmaps illustrate the learning progress of a CNN student model on five learning concepts: "Fish", "Insects", "Large carnivores", "Flowers", and "Vehicles", over the CIFAR-100 dataset.

We observe two main findings from these heatmaps. First, the progress of validation accuracy with the KADT method in the bottom heatmap is more stable and evenly distributed over the five learning concepts than the one with the L2T method in the top heatmap. For example, the final episodes 288-344 over the "Insects" and "Flowers" learning concepts are more evenly distributed for the KADT method in comparison to the L2T method. Second, the student model with the L2T method suffers from forgetting, which can be observed around episodes 144-160 in the "Fish" learning concept and around episodes 208-232 in the "Flowers" learning concept. These findings confirm the effectiveness of learning the performance of a student model on multiple learning concepts in the KADT method, compared with learning through a concept-agnostic way in the L2T method.

6.5.4 Evaluating Teaching Strategies

In this experiment, we compare the training accuracy of the student models achieved by the KADT and L2T methods since the L2T method is the state-of-the-art reinforcement learning-based teaching model.

Figure 6.6 shows the average training accuracy curve of DKT [137] as a student model on the ASSISTments2009 dataset, SVM model on the IMDB dataset, LSTM model on the MovieLens dataset, and MLP model on the CIFAR-100 dataset. We compare the results from the KADT method (blue line) with the ones from the L2T method (red line). It can be observed that the student's training accuracy curves with the KADT method are more stable and converge faster than the ones with the L2T method while achieving a higher training accuracy

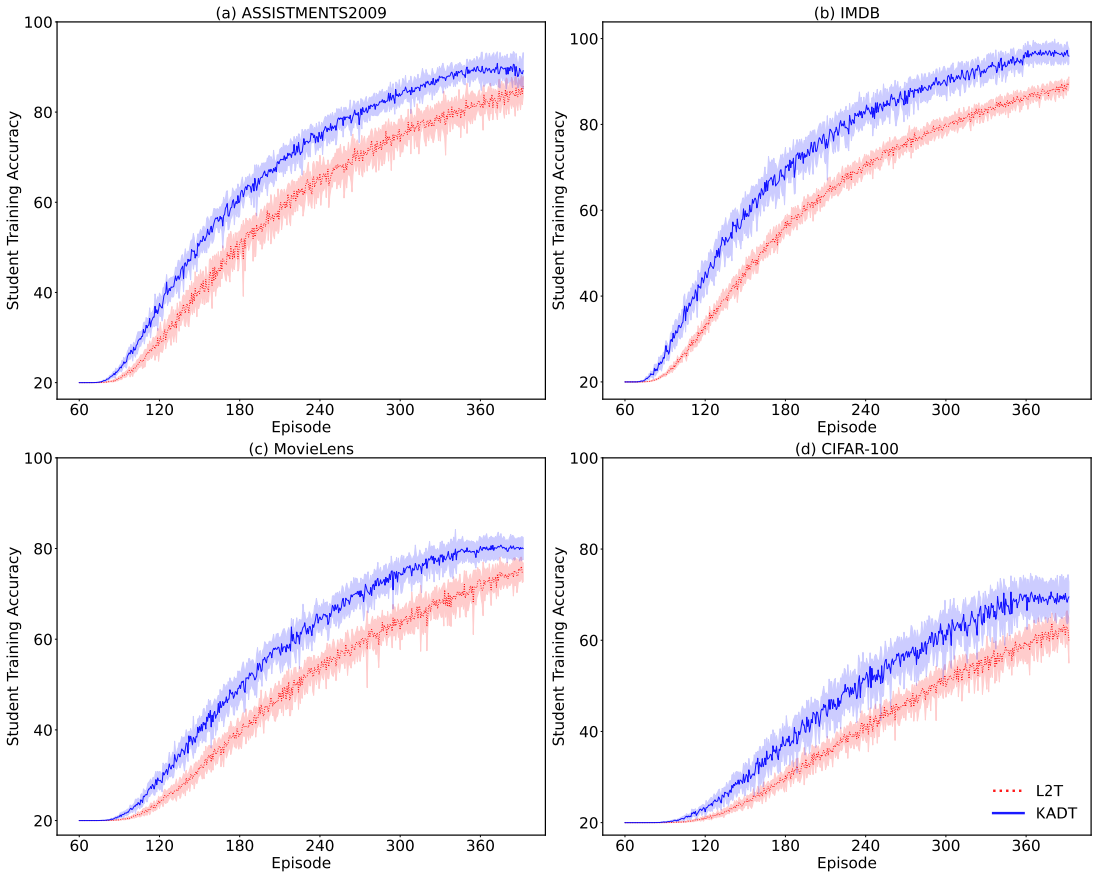


Figure 6.6: Student training accuracy curves averaged over five runs by our KADT method in comparison with the L2T method. (a) DKT student model on the ASSISTments2009 dataset. (b) SVM student model on the IMDB dataset. (c) LSTM student model on the MovieLens dataset. (d) MLP student model on the CIFAR-100 dataset.

value after convergence. These findings reflect that a data teaching strategy evolved by the KADT method had a better impact on the student's training performance in comparison to the L2T one.

To understand the behavior of each teaching method, we analyze their student's training wall-clock time during *Phase 2* in the training process (See Section 6.4.4.2). Figure 6.7 shows a wall-clock training time comparing our model with the other baselines using the CNN student model on the CIFAR-100 dataset. This analysis was executed on an NVIDIA RTX 2080Ti GPU with 11GB of RAM. We observe that the KADT model yields a better wall-clock training time reduction than the other baselines. More specifically, KADT achieved a faster convergence with 7% margin above the nearest comparative (L2T) at wall-clock time 7000.

We asymptotically calculate the computational complexity of each teaching model. Table 6.3 summarizes the test-time (i.e., no back-propagation) computational complexity for each model. For the RandomTeach model, it is basically a uniform sampling operation for a mini-batch of size m; thus, its computational complexity is $O(m)$. For the SPL model, it works through k sample selection iterations; each iterates over the whole samples s in a

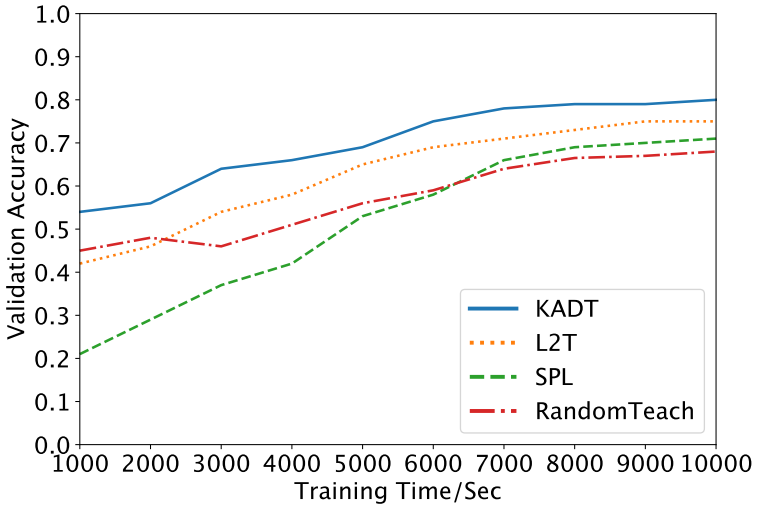


Figure 6.7: Comparing different teaching strategies for training wall-clock time analysis using a CNN student on the CIFAR-100 dataset.

Table 6.3: Computational complexity for each teaching model in our experimental setup.

Method	Time Complexity
RandomTeach	$O(m)$
SPL	$O(ksd^2)$
L2T	$O(m + 2dnl)$
KADT (ours)	$O(2md + 2dnl)$

training dataset performing a matrix multiplication over the input dimensions d; thus, its total complexity is $O(ksd^2)$. For the L2T model, it first utilizes a uniform random sampling to get a mini-batch with size m; then, it performs two forward passes, one for the actor and another for the critic networks of the DDPG algorithm. The computational complexity for a forward pass for a feed-forward neural network is $O(dnl)$, where d is the input dimensionality, n is the number of neurons per layer, and l is the number of layers. Thus, the total computational complexity for the L2T model is $O(m + 2dnl)$. Finally, the KADT model utilizes a KT model to estimate the state, which works on the last sampled mini-batch with size m to estimate a knowledge state over the defined learning concepts. The computational complexity of our memory-augmented knowledge tracing model is $O(2md)$, where m is the size of the mini-batch, and d is the number of dimensions for each sample. We refer the reader to [2] for further details on driving the KT model computational complexity. Besides, KADT follows a similar forward pass to the L2T using the DDPG algorithm. Thus, the final computational complexity for KADT is $O(2md + 2dnl)$.

6.5.5 Ablation Study

We conduct an ablation study to assess the effect of each individual component in our KADT method. We consider and compare two different variants, including 1) a baseline variant

Table 6.4: The Average AUC results for comparing different variants of KADT and L2T over all the datasets.

Model	Components			Datasets			
	RL	KT	Attention	ASSISTments2009	IMDB	MovieLens	CIFAR-100
L2T	✓	×	×	84.31 ± 0.03	89.46 ± 0.06	77.23 ± 0.03	65.33 ± 0.06
KADT-Basic	✓	×	×	84.69 ± 0.01	89.83 ± 0.03	77.70 ± 0.04	65.87 ± 0.03
KADT-KT	✓	✓	×	85.79 ± 0.02	90.84 ± 0.04	78.90 ± 0.03	67.11 ± 0.02
KADT	✓	✓	✓	**87.10 ± 0.04**	**92.24 ± 0.02**	**80.31 ± 0.02**	**68.75 ± 0.03**

called "KADT-Basic", which follows the same definitions of state and action from the L2T model [41] but uses our reward function design, and 2) a variant with the KT component working with mean pooling, i.e., taking the mean of knowledge representation vectors for samples within the same class label as the knowledge representation for the class label while treating them as equally likely, we call it "KADT-KT". Our complete model KADT has both KT and attentive pooling components. We perform multiple independent runs and calculate the statistical significance of the results using a student t-test, counting a p-value < 0.05 as a statistically significant finding.

Table 6.4 summarizes the average AUC results for different variants in the ablation study. We highlight the findings as follows. Firstly, the impact of our reward function design can be observed by comparing the KADT-Basic variant with the L2T model [41]. A significant performance margin of 0.38, 0.37, 0.47, and 0.54 exists between these two models for *ASSISTments2009*, *MDB*, *MovieLens*, and *CIFAR-100*, respectively. Secondly, the impact of the KT model can be observed by comparing the KADT-Basic variant with the KADT-KT variant. A statistically significant performance enhancement by a margin of 1.10, 1.01, 1.20, and 1.24 is achieved for *ASSISTments2009*, *MovieLens*, and *CIFAR-100*, respectively. Finally, the impact of the attentive pooling technique can be observed by comparing the KADT-KT variant with the complete KADT model. A statistically significant performance enhancement by a margin of 1.31, 1.40, 1.41, and 1.64 is achieved for *ASSISTments2009*, *MovieLens*, and *CIFAR-100*, respectively.

6.6 Summary

In this chapter, we proposed a novel data teaching method, called Knowledge-Augmented Data Teaching (KADT), which dynamically learns knowledge representations of a student machine learning model over multiple learning concepts in a supervised learning task while optimizing a data teaching strategy to maximize the student's performance. KADT resolved several limitations of the existing RL-based data teaching methods. First, it eliminates the need for manually designing teaching state representations across different learning tasks by learning it using a knowledge tracing model. Second, it proposes a dense formulation for a teaching reward function that provides more training signals to an teaching agent during training. Third, it provides an efficient attention mechanism to condition the action sampling on training data. Lastly, the framework of the KADT model is agnostic to the student model's architecture or size given the data on its loss function distribution over training data; thus, it could be easily

applied to recent state-of-the-art machine learning models with minimal effort. We performed a comprehensive evaluation comparing KADT with the state-of-the-art data teaching methods on different supervised machine learning tasks. The experimental results not only showed that KADT significantly outperformed other comparatives but also was able to generalize across different tasks with minimal calibration.

Chapter 7

Addressing Limitations in Existent Knowledge Tracing Datasets

7.1 Overview

Despite of the availability of various KT datasets [42, 83, 23, 137, 27, 160], there are limitations in these datasets when evaluating advanced deep KT models, including (1) missing text data for question tags and description text for learning concepts, which limits the use of deep language models that can find effective question embedding representations, (2) missing ground-truth on the relationships among learning concepts, which can be used by deep graph neural networks to incorporate the influence between learning concepts when updating a knowledge state, (3) missing ground-truth on question difficulty, which can provide an auxiliary signal when predicting the likelihood of correct answer, (4) lack of diversity in student groups as the majority of these datasets were collected from school-grade students, (5) lack of insights on a student's uncertainty while answering a given question, which can provide features to quantify the likelihood of important answer aspects such as forgetting, guessing, or slip (i.e., mistaken answer). Table 7.1 compares existing KT datasets and our proposed dataset on several key aspects that highlight the mentioned limitations.

To address these limitations, we propose a new dataset named *DBE-KT22*, which was collected from a course on relational databases taught at the Australian National University (ANU) in Australia for undergraduate and graduate students across multiple disciplines, including computer science, engineering, arts, and business between 2018-2021 academic years. The scientific merits of the DBE-KT22 dataset can be summarized as follows:

- Providing detailed meta-data for questions, including tag text, hints text, choice text, and images that provide multi-model feature sources for question embedding representation learning.
- Providing detailed meta-data for the involved learning concepts, including learning concept tag text and description text to facilitate learning representative learning concept embeddings.
- Incorporating the ground truth on the relationships between learning concepts and each other and between learning concepts and questions in the form of graphs to facilitate the

Table 7.1: Comparing essential aspects for KT datasets.

Dataset	Question Text	Relationships		Question Difficulty	Answer Confidence
		Question - KC	KC - KC		
ASSISTments	×	✓	×	✓	×
STATICS	×	✓	×	×	×
Junyi Academy	×	✓	×	✓	×
Simulated-5	×	✓	×	×	×
KDDcup	×	✓	×	×	×
EdNet	×	✓	×	×	×
Eedi	✓	✓	×	×	×
DBE-KT22	✓	✓	✓	✓	✓

use of graph representation learning methods.

- Presenting the ground truth on question difficulty provided by domain experts teaching the subject over a long time period. This provides auxiliary features that could be used for answer prediction and evaluating methods targeting quantifying the question difficulty, such as question recommendation and curriculum generation models.

- Capturing different aspects that reflect a student's uncertainty while answering questions, including self-feedback on question difficulty, self-feedback on confidence in the answer, indication for using hints, the number of times a student changes their answer, and the total time taken to answer. As difficulty rating might vary across experts [57], we record a student's feedback during exercise answering to highlight further any variance in preserving questions difficulty. This dataset enables the use of answer uncertainty features in addition to the features from the previous question answering sequence.

The DBE-KT22 dataset is publicly available through the Australian Data Archive (ADA) repository: https://dataverse.ada.edu.au/dataset.xhtml?persistentId=doi:10.26193/6DZWOH.

The remainder of this chapter is organized as follows. Section 7.2 introduces the details of the proposed DBE-KT22 dataset, including data collection, distribution, and formatting. Section 7.3 presents our experimental analysis to highlight the characteristics of the proposed dataset. Finally, Section 7.4 concludes the work.

7.2 The DBE-KT22 Dataset

In this section, we introduce the generation procedure of the DBE-KT22 dataset, including data collection, data relational schema, data distribution, privacy preservation, and data sharing. Additionally, we present the key statistical characteristics of the dataset and visualize them using graphs. Figure 7.1 presents the generation procedure of the DBE-KT22 dataset. In the data collection step, we elaborate on the means to collect the data, the types of collected data,

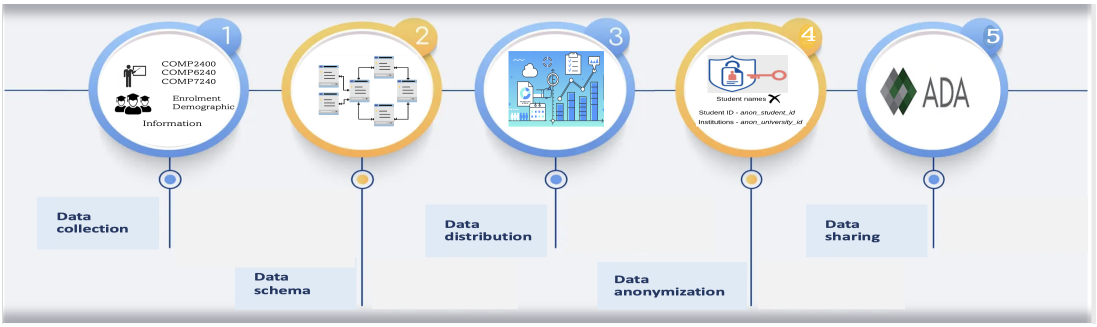


Figure 7.1: The DBE-KT22 dataset production steps.

and aspects of the participation environment. The structure of the collected data and their relational dependencies is introduced in the data schema step. The data distribution step presents statistical aspects of the collected data. We clarify the workflow to preserve participants' privacy in the data anonymization step. Finally, we illustrate the steps for sharing our collected data and making it publicly available in the data sharing step.

7.2.1 Data Collection

The DBE-KT22 dataset was collected based on real student exercise practicing in the *Relational Databases* course taught at the Australian National University (ANU) in Australia. Students were mainly at the undergraduate level from different specializations, including computer science, engineering, science, business, economics, arts, social sciences, and law and humanities. We note that to the best of our knowledge, our dataset is the first to cover such broad undergraduate student groups across the available KT datasets. The data was collected over a three-year time period from 2018 to 2021.

We utilized an online exercise practicing platform developed by the ANU named the *Database Bench* platform [1]. *Database Bench* is a web-based application that can be accessed exclusively by ANU students and staff using their university IDs. The platform enables students to practice exercises self-paced, where exercises are organized by study weeks. Figure 7.2 shows screenshots of the main web pages in the platform. Firstly, a student would log in using their university credentials. Then, they are redirected to a home page showing weekly exercise sets for the course; for each set, they can track their completion progress shown through a progress bar UI. Once a student selects a specific weekly exercise set, they see all the exercises within it and can select a given exercise to practice. On the exercise practice page, the student is presented with question's data including title, illustrative details such as graphs, tables, or images, and answer choices. We note that all of the questions in the dataset are multi-choice ones. Besides, a student can see a hint if the instructor provided it for the current question. To record a student's perspective on question difficulty and their trust in the answer before submitting it, we provide two optional questions under each exercise asking the student for their feedback on difficulty and trust in the selected answer. Moreover, we passively record the number of times a student changed their answer selection and the total

[1] https://cs.anu.edu.au/dab/index.html

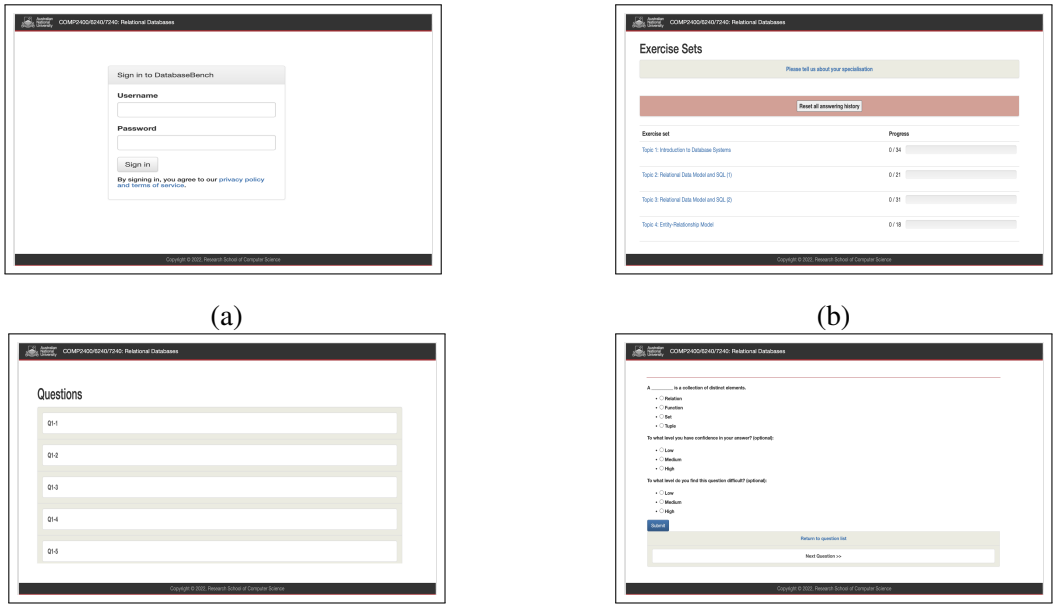


Figure 7.2: Screenshots for the *Database Bench* web platform for exercise answering. (a) login page. (b) weekly exercise set page. (c) questions list per exercise set. (d) question answering page.

time taken to submit the answer for additional features about their confidence in the answer.

Figure 7.3 depicts a workflow for collecting the exercise answering data, including 1) acquiring university credentials upon enrollment in the course, 2) logging into the *Database Bench* web platform, 3) in case it was the first time to log in, answering a questionnaire about specialization and confirming consent on the course practicing code of conduct, 4) practice exercises, and 5) save practice activity data into the database. We collected practice activity data over three academic years in the period $[2019 - 2021]$ and ordered the answering sequence for each student chronologically. The schema of the collected data is introduced in the following section.

7.2.2 Data Schema

The schema for the DBE-KT22 dataset is presented as a set of entities such as questions, students, and learning concepts and their relationships. Figure 7.4 shows the entity-relationship diagram (ERD) for the DBE-KT22 database.

We describe the relational schemas for the entity-relationship diagram depicted in Figure 7.4 in the following:

- **dbe_question**: this is the main table in this database as it contains all question meta-data.
- **dbe_choice**: this table contains meta-data for choices per each question.
- **dbe_hints**: this table stores hint data for questions.

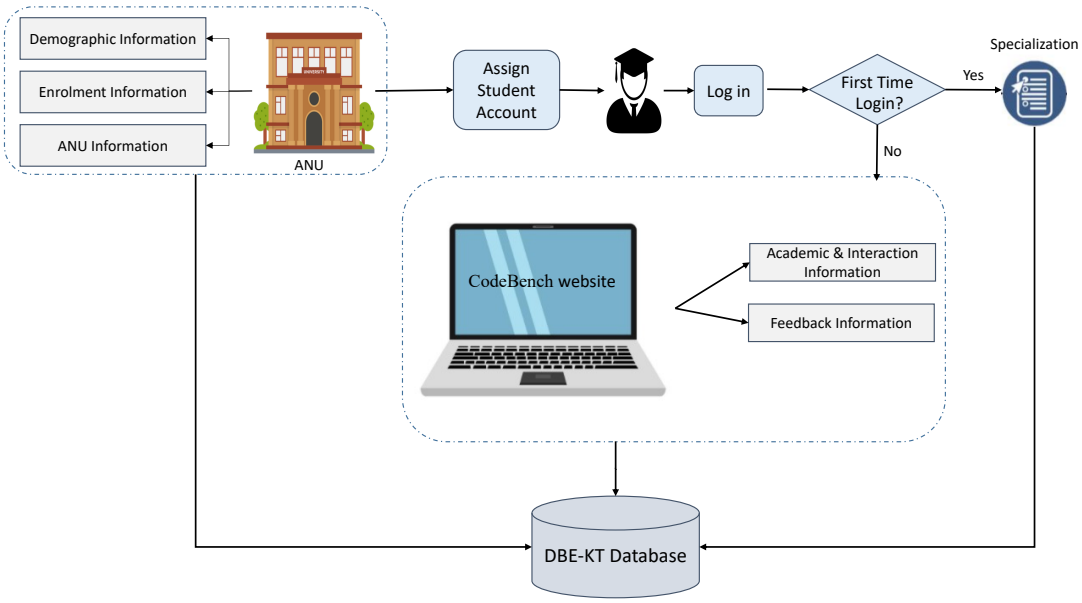


Figure 7.3: DBE-KT22 data collection workflow.

- **auth_user**: this table contains user login credentials and contact information such as email.
- **dbe_transaction** this table records question answering transaction data including the answer state.
- **dbe_week**: this table contains weekly exercise sets, which are groups of exercises based on a weekly decomposition for the course period.
- **dbe_consentform**: this table stores the data of the student consent including specialization inquiry and confirmation on the course policy.
- **dbe_specialization**: this is a lookup table for specialization data.
- **dbe_knowledgecomponent**: this table contains meta-data for learning concepts (KCs) in the course.
- **dbe_knowledgecomponents_dependendts**: this table stores relationships between the KCs.
- **dbe_question_kcs**: this table stores relationships between the questions and KCs.

We decomposed the previously described ERD into a set of files to facilitate data sharing and distribution. We followed the Comma Separated Value (CSV) file format for our dataset files for its popularity and the availability of processing software packages. We name each file with the same name as its equivalent table from the ERD.

Besides, we provide a Python script (i.e., uploaded with the dataset files) to generate training question answering sequences. The script takes three parameters, including the desired

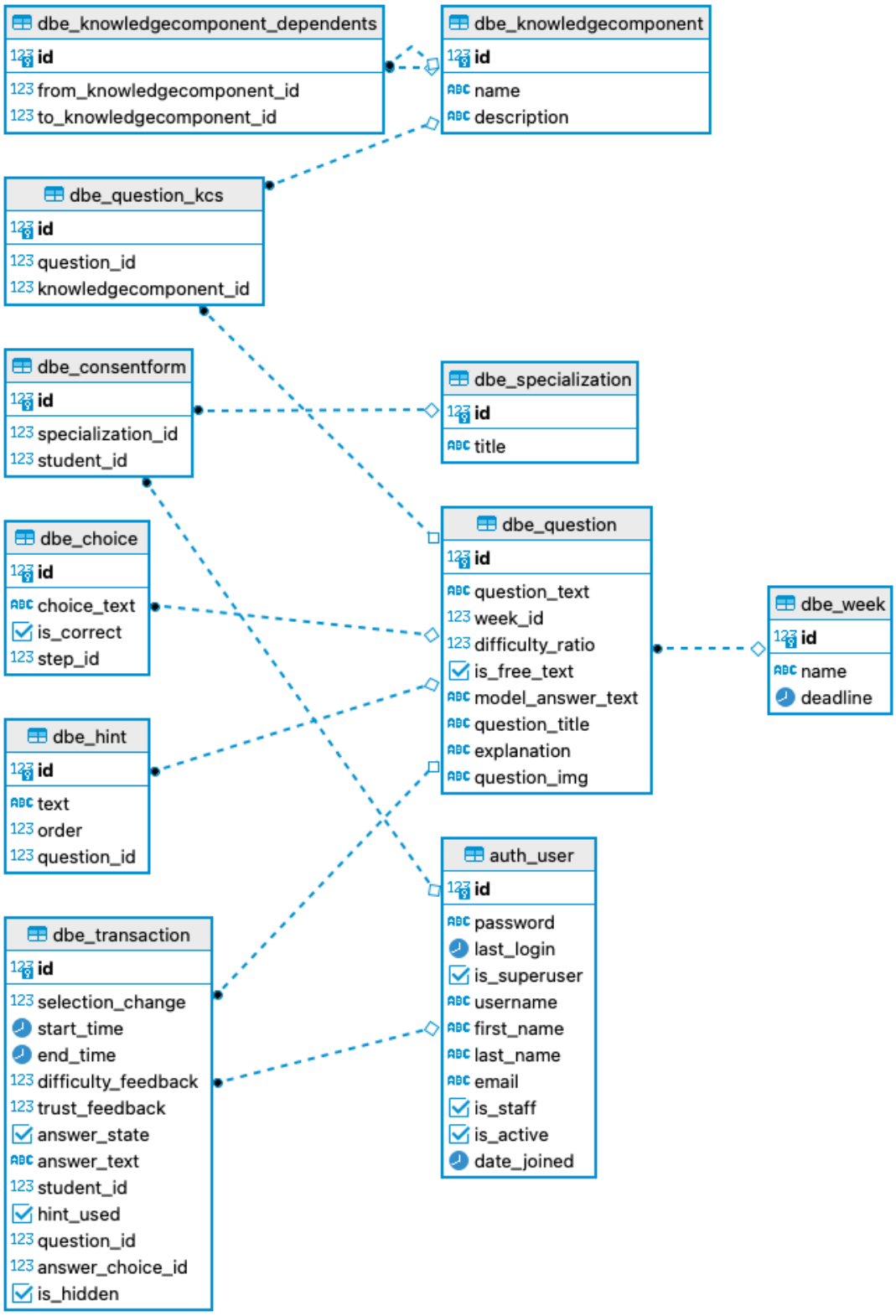


Figure 7.4: Entity-Relationship (ER) diagram for the DBE-KT22 database.

sequence length for sequencing a student answering history, a padding character for padding sequences shorter than the length argument, and an output file path to save the result. The resultant sequences file named *practice_sequences.json* is formatted using the JavaScript Object Notation (JSON) format. The file is structured as an array of JSON objects; each has the following fields:

- **student_id**: ID of the current student in the sequence.
- **seq_len**: the length of the current sequence sample. Note that in the case of a sequence with a shorter length than the length argument, this will show the sequence length without counting the padding chars.
- **question_ids**: unique ids for questions in the sequence.
- **answers**: answer status for each question in the sequence with 0 for wrong and 1 for correct status.
- **gt_difficulty**: ground truth difficulty level for each question provided by the course instructors. We use 1 for easy, 2 for medium, and 3 for difficult.
- **difficulty_feedback**: student's feedback on question difficulty before submitting the answer. We use 0 for not provided, 1 for easy, 2 for medium, and 3 for difficult.
- **answer_confidence**: student's feedback on their confidence in the selected answer. We use 0 for not provided, 1 for low, 2 for medium, and 3 for high.
- **hint_used**: a binary indicator for using hint per each question in the sequence.
- **time_taken**: time in seconds taken to answer each question in the sequence.
- **num_ans_changes**: count for answer selection change per each question in the sequence.

7.2.3 Dataset Distribution

In this section, we investigate the data distribution in the DBE-KT22 dataset using graphs. We start by showing the distribution of students across specializations. Note that this only includes students who responded to the specialization questionnaire in the consent form. Figure 7.5 shows a bar chart to visualize this distribution. The majority of participants come from engineering and computer science specializations. In Figure 7.6, we show a density plot for the distribution of relevant KCs per a given question. It can be observed that the majority of questions in the dataset has 1 to 2 relevant KCs. As we include question text in our dataset, it is useful to show the text token length distribution as it is a vital argument for text embedding models. Figure 7.7 shows a density plot for the token length in the question text. The majority of questions has a length of less than 50 tokens, with a minority stretching this number around 300 tokens. Figure 7.8.a shows a pie chart for the distribution of question difficulty in the dataset that reflects a reasonable balance between the easy and medium levels and a minority of difficult questions. As per Figure 7.8.b, the majority of questions (around 85%) in the

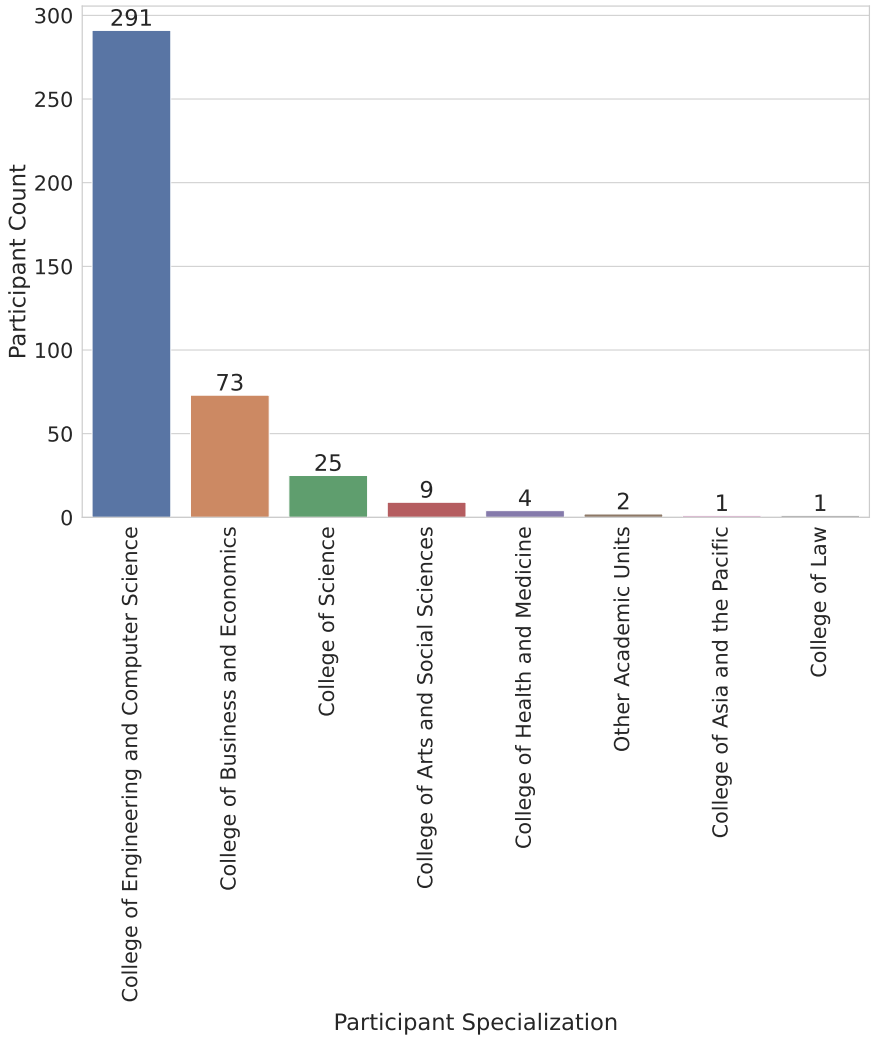


Figure 7.5: The distribution of student specialization in the DBE-KT22 dataset.

dataset does not have an explanation (i.e., instructor's description of the modal answer), which is aligned with the question difficulty distribution as 84% of questions are easy or medium while usually only difficult ones need to have an explanation. The hint (i.e., a small piece of information that could help to approach the answer, yet it couldn't be used as an answer) distribution across questions in Figure 7.8.c is showing that around 66.5% of questions has a hint note to help the student when requested, which covers all the medium and difficult questions in the dataset.

7.2.4 Data Anonymization

While our dataset collects data based on student exercise practice activities, their personal data is usually unrelated to the knowledge tracing context. Thus, we did not aim to collect any personal identifying information in our dataset. We only included two fields per each student

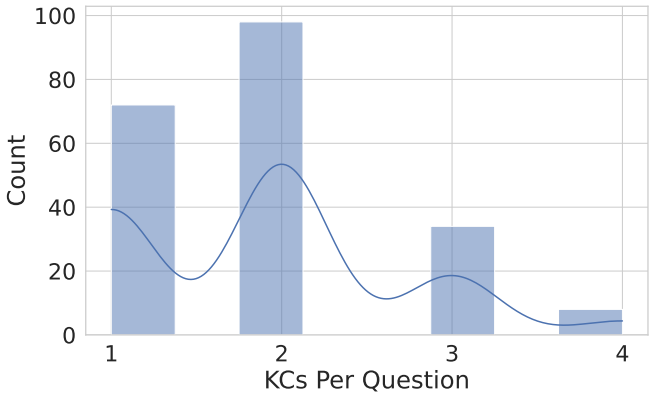


Figure 7.6: The distribution of the number of KCs per question in the DBE-KT22 dataset.

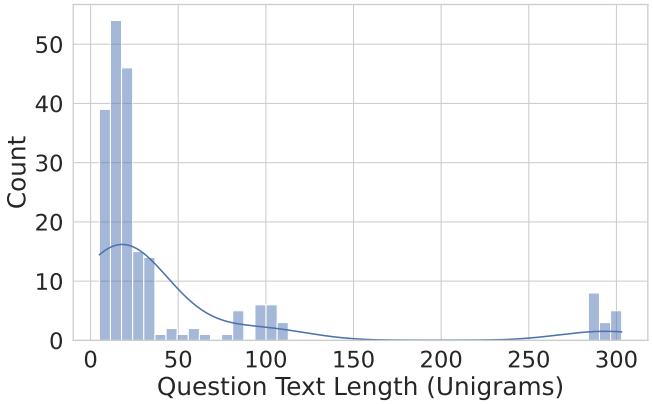
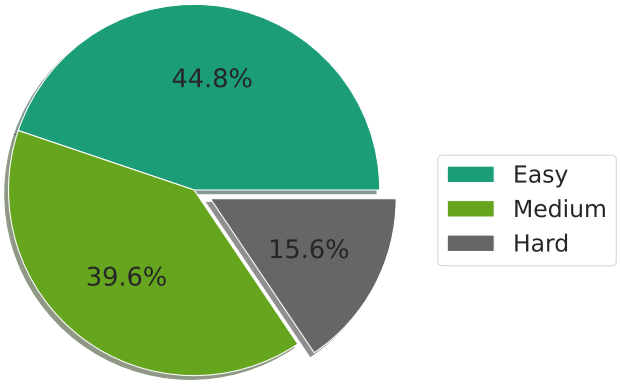
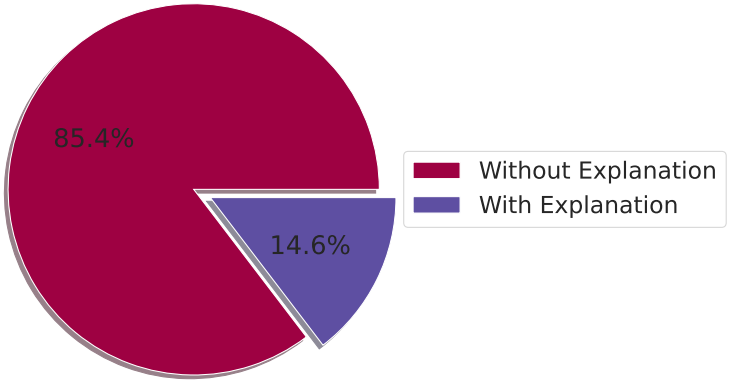


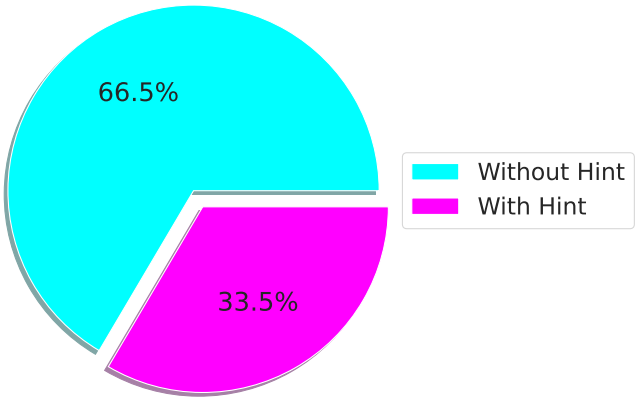
Figure 7.7: The distribution of question text length in the DBE-KT22 dataset.



(a)



(b)



(c)

Figure 7.8: Distribution of question's difficulties, the existence of explanation, and the existence of hint in the DBE-KT22 dataset. (a) difficulty distribution. (b) explanation distribution. (c) hint distribution.

record, including their ID and specialization. For the IDs, we use an incremental identifier for the dataset records rather than the university-issued ID number to preserve students' privacy. While the specialization field was an optional one during the data collection process, and it can not be used to backtrack students' identities.

Our data collection and privacy preservation procedures have been reviewed and approved by the ANU human research ethics committee [1] under the protocol number 2017/543.

7.2.5 Data Sharing

We share the DBE-KT22 dataset files through the *Australian Data Archive* (ADA) platform [2] under an unrestricted access policy. ADA is a platform maintained by the ANU for sharing scientific data with the public, and it provides advanced data indexing and search capabilities. Our dataset can be downloaded through its relevant page [3] on the ADA that includes general information describing its content, license, and research objectives. The uploaded DBE-KT22 dataset content includes the following files:

- **kc_relationships.csv**: file contains relationships among KCs.
- **kcs.csv**: file contains meta-data of KCs.
- **question_choices.csv**: file contains choices meta-data per each question.
- **question_kc_relationships.csv**: file contains relationships between questions and KCs.
- **questions.csv**: file contains meta-data for questions including hint and explanation texts.
- **transaction.csv**: file contains meta-data for question practice attempts.
- **specialization.csv**: file contains meta-data for specializations.
- **student_specialization.csv**: file contains lookup data linking student ids with their corresponding specializations.
- **sequencer.py**: Python script file for generating training sequences.
- **practice_sequences.json**: file contains generated training question answering sequences after executing the provided sequencer Python script.

7.3 Experiments

In this section, we introduce our experimental study of the DBE-KT22 dataset. We conduct two experiments including an exploratory data analysis (EDA) experiment and a question representation learning experiment. The former explores various dataset aspects relevant to the answer prediction task, while the latter evaluates different ways for effective question representation learning.

[1] human.ethics.officer@anu.edu.au
[2] https://ada.edu.au/
[3] https://dataverse.ada.edu.au/dataset.xhtml?persistentId=doi:10.26193/6DZWOH

7.3.1 Exploratory Data Analysis

In this experiment, we aim to explore the characteristics of the DBE-KT22 dataset by analyzing the recorded data. We outline six questions investigating the independent variables impacting a student's answer status in the data. Understanding the relationships between these variables and the answer status is vital for any KT usage scenario. The questions are listed as follows.

- **Q1**: How do the question difficulty levels affect the answers of students?
- **Q2**: How does students' answer confidence feedback relate to their answers?
- **Q3**: How well is students' difficulty feedback aligned with the ground truth difficulty of questions?
- **Q4**: How well is students' answer confidence feedback aligned with the ground truth difficulty of questions?
- **Q5**: How dependable is the answering time as an indicator factor for the answers of students?
- **Q6**: How dependable is the hint usage as an indicator factor for the answers of students?

To answer **Q1**, Figure 7.9 shows a grouped bar chart for answer status per each ground truth (i.e., provided by the course instructors) question difficulty level. It can be noticed that the distribution between true and false statuses is moving from being imbalanced towards the true status in the easy and medium difficulty levels to a balanced one in the hard level, reflecting the increase of knowledge uncertainty with the increase of question difficulty. To answer **Q2**, we show the distribution of answer status over student's confidence feedback in Figure 7.10, which shows an increasing linear pattern in the true answer probability w.r.t the increase in the confidence level. Answering **Q3**, we investigate the distribution of student's difficulty feedback over the ground truth question difficulty levels. As per Figure 7.11, one can notice a linear decrease pattern in the number of students reporting an easy question difficulty (blue bars) with the increase of question ground truth difficulty level, which supports the quality of the ground truth labeling done by course instructors. Similarly, to answer **Q4**, we inspect the distribution of a student's answer confidence feedback over the ground truth question difficulty. As per Figure 7.12, we observe that the percentage of high confidence feedback (green bars) linearly decreases with the increase of ground truth question difficulty. For answering **Q5**, we show in Figure 7.13 the distribution of answering time (i.e., time in seconds taken till submitting the answer) over answer status and ground truth question difficulty levels. We observe that for the easy and medium difficulty levels, the differences between the second quartiles (median) and third quartiles of answering time are significant, with more time taken for false (i.e., wrong) answer status, while these differences are less significant in the hard question difficulty level for the increased uncertainty. Finally, to answer **Q6**, we show the answer status distribution over hint usage binary indicator in Figure 7.14. We calculated each bar's percentage of wrong answers (i.e., false status). We observe that using the hint is an effective indicator of a gap in

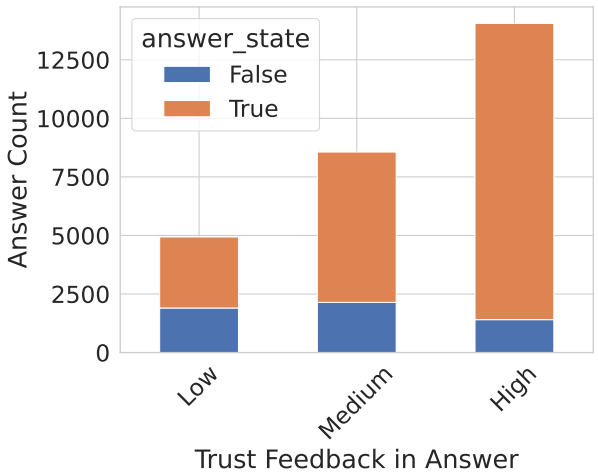


Figure 7.10: A bar chart implying the distribution of answer status over the student's confidence feedback.

a student's knowledge and would probably imply a wrong answer status, as the percentage of wrong answers is higher when using the hint compared to not using it.

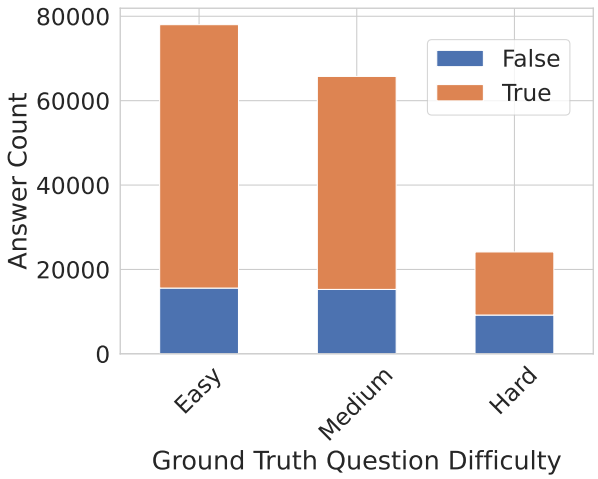


Figure 7.9: A bar chart implying the impact of ground truth question difficulty on the answer status.

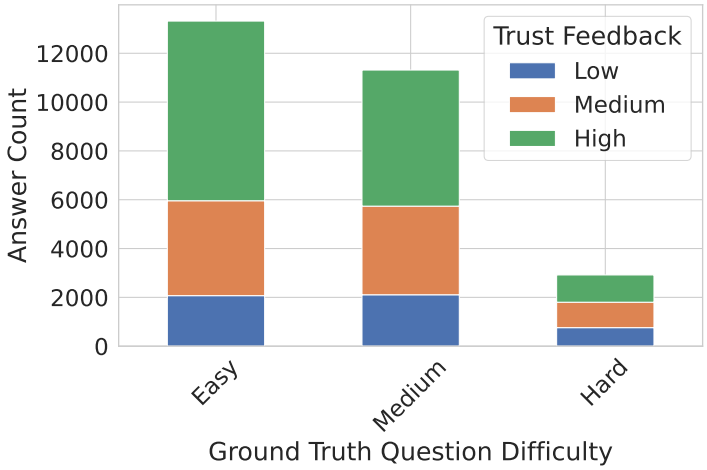


Figure 7.12: A bar chart showing the distribution of student's answer confidence feedback over ground truth question difficulty.

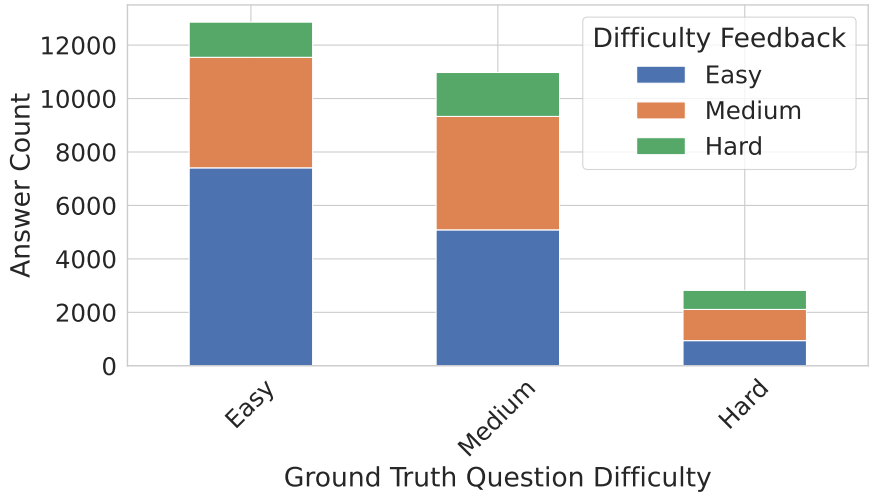


Figure 7.11: A bar chart showing the distribution of student's difficulty feedback over ground truth question difficulty.

7.3.2 Question Representation Learning

In this experiment, we evaluate different ways of learning a question representation using question text data. We utilize the uncased-base pre-trained BERT [37] language model using *Transformers* NLP models hub [1]. We define the following questions to guide our evaluation of this experiment:

- **R1**: What are the different ways to distill question embedding from a pre-trained BERT language model? How effective is each way in clustering relevant questions?

[1] https://huggingface.co/bert-base-uncased

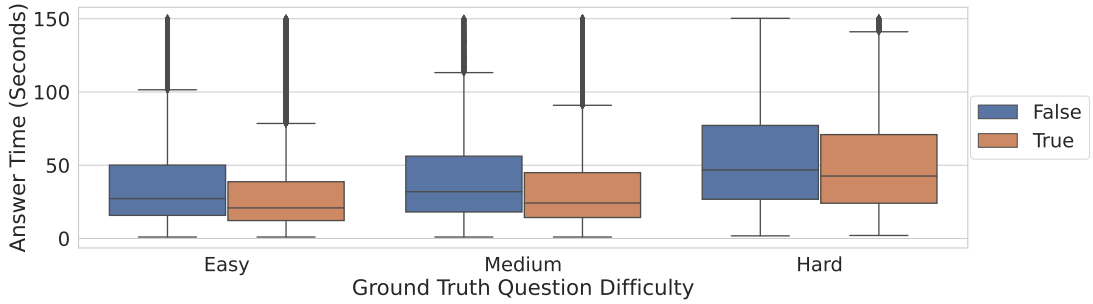


Figure 7.13: A grouped boxplot chart showing the distribution of answering time across answer status and question difficulty levels.

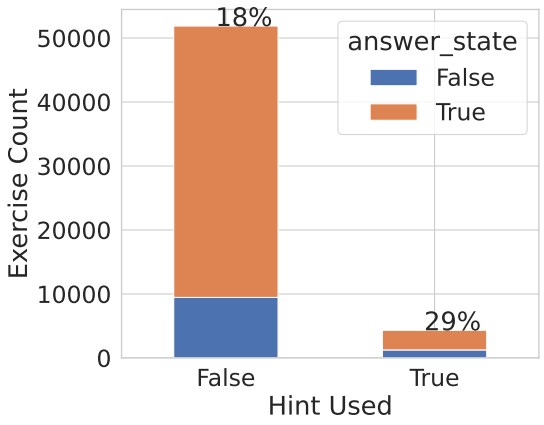


Figure 7.14: A bar plot showing answer status distribution over hint usage. The percentage of wrong answers out of the total answers is shown above each bar.

- **R2**: How to fine-tune pre-trained embedding representation of question text? What is the effect of fine-tuning dataset size on the quality of embedding representation?

To answer **R1**, we need a reference ground truth embedding representation for questions to evaluate different ways of distilling pre-trained text embeddings. We use the ground truth data on relationships between questions and KCs to design such a reference representation. We represent the ground truth on the similarity between a question-question pair by the common KCs between them. Thus, a question in the reference ground truth representation is represented with a binary vector of length N (for the total number of KCs in the dataset), with 1s in the positions of its relevant KCs and 0s elsewhere. For distilling question text embedding from the pre-trained BERT model, we evaluate five different ways including:

1. **CLS token** embedding: represents each question with the CLS token of the final layer in the BERT model.

2. **last hidden layer** embedding: represents each question with the CLS token of the last hidden layer in the BERT model.

3. **last second hidden layer** embedding: represents each question with the CLS token of the last second hidden layer in the BERT model.

4. **last third hidden layer** embedding: represents each question with the CLS token of the last third hidden layer in the BERT model.

5. **pooled mean** embedding: represents each question with the reduced mean of the CLS tokens of the last three hidden layers in the BERT model.

6. **pooled max** embedding: represents each question with the reduced max of the CLS tokens of the last three hidden layers in the BERT model.

We use the K-Means clustering algorithm [69] for performing clustering over each embedding method in the evaluation. To decide on the number of clusters K, we depend on a data-informed approach using the Elbow method [170]. Figure 7.15 shows a curve for the error sum of squares (SSE) per each K value. SSE is calculated by getting the sum of squared differences between each point and its' cluster mean point. We select $K = 10$ as a good configuration based on the SSE curve. In order to visualize the clustering results per each method, we apply t-SNE manifolding [174] on each embedding variant using two manifold components. Figure 7.16 depicts the clustering results for each embedding method, we found that the **last hidden layer** embedding method outperformed others (see Table 7.2) on the defined clustering performance metrics including the *Inertia* metric [69], and the *Homogeneity* metric [145]. The *Inertia* metric measures clustering quality by calculating the distance between each data point and its cluster's centroid, squaring this distance, and taking the sum of squares for each cluster. The less the value of this metric, the better the clustering quality. The *Homogeneity* metric measures the alignment of class labels within a given cluster using a ground truth of class labels. It has a value in the range of $[0, 1]$ with 1 for a perfect alignment and 0 for a complete

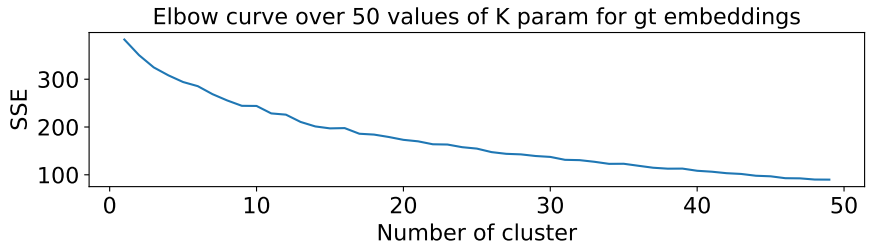


Figure 7.15: The Elbow curve comparing the error sum of squares (SSE) over the number of clusters K using the ground truth question similarity representation.

Table 7.2: Summary for clustering performance results for ground truth embedding and different methods of question text embedding using Inertia and Homogeneity metrics.

Embedding Method	Inertia	Homogeneity
Ground Truth Embedding	203.8	1.0
CLS token embedding	1923.2	0.63
last hidden layer embedding	**1908.3**	**0.68**
last second hidden layer embedding	1935.0	0.59
last third hidden layer embedding	2045.6	0.58
pooled mean embedding	1981.7	0.55
pooled max embedding	2063.0	0.58

dis-alignment. We use the cluster labels generated by the ground truth question representation as class labels to calculate this metric. The higher the value of this metric, the better the clustering quality.

In order to answer **R2**, we investigate fine-tuning the best question text embedding method from **R1** evaluation. We mean by fine-tuning to adapt the embedding parameters of the pre-trained BERT model on our question text data by formulating a supervised learning task to predict the number of shared KCs between a pair of questions using their text embeddings as input. To show the impact of fine-tuning dataset size on performance, we configure three size settings including 10%, 25%, and 50% of the original training dataset size. This enables us to reflect on the expected enhancement in the learned embedding with more ground truth data about question-KC relationships introduced. We compare the performance of each variant using the same clustering metrics used in **R1** evaluation including *Inertia* and *Homogeneity* metric. Furthermore, we compare the performance of the fine-tuned embedding variants on classifying question ground truth difficulty label (i.e., $label \in \{1, 2, 3\}$) from text embedding using classification accuracy metric. As summarized in Table 7.3, we observe a linear increasing enhancement pattern on the performance of fine-tuned embedding with more ground truth data introduced, yet using 10% of the ground truth data size was sufficient to get a significant enhancement with a magnitude of -37.8, and 0.03 for the *Inertia* and *Homogeneity* metrics respectively comparing to the best performer from **R1** evaluation. For question difficulty classification results shown in Figure 7.17, we notice a similar linearly increasing pattern in the

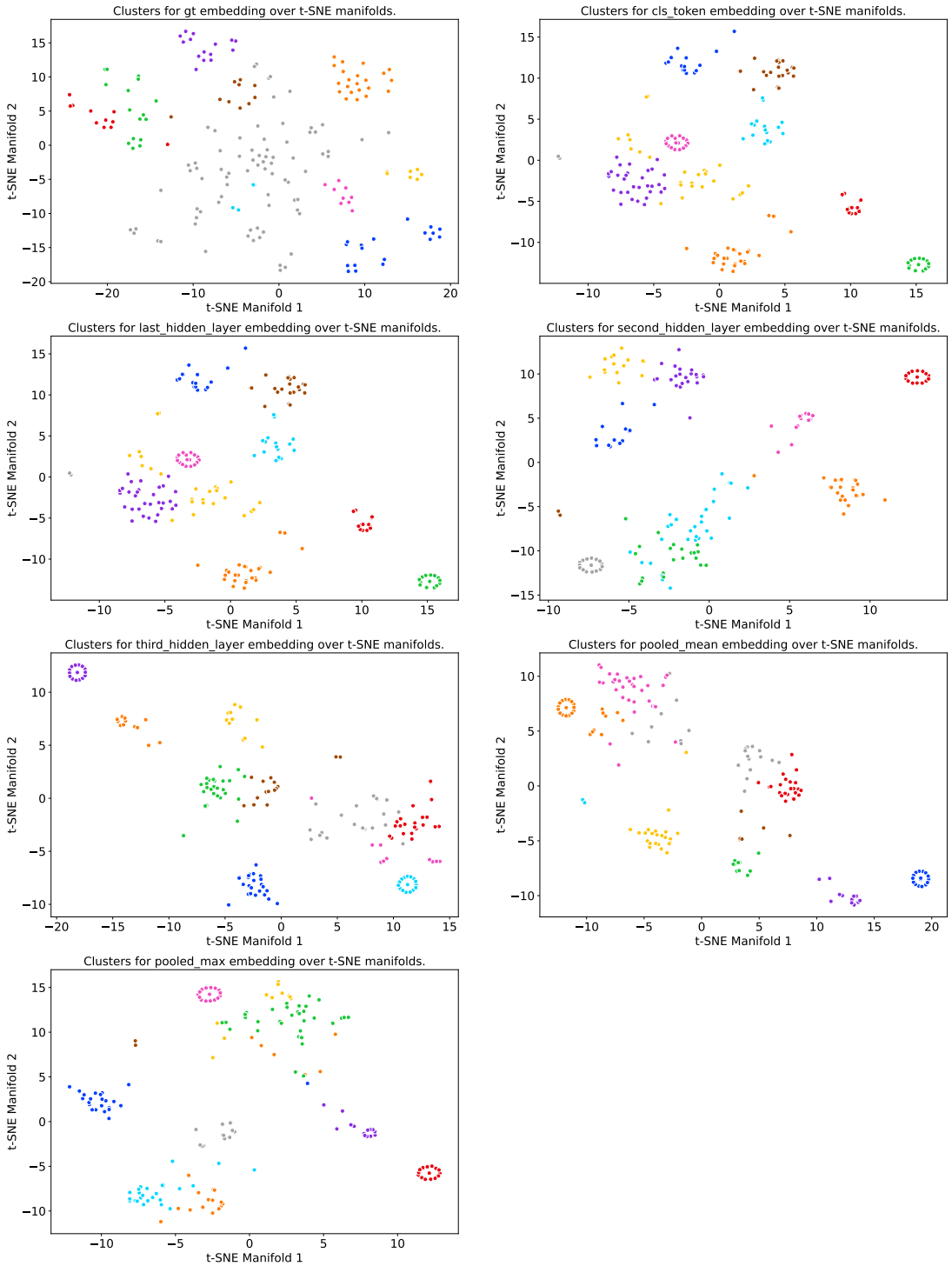


Figure 7.16: A t-SNE two manifold visualization of clustering result for each question embedding method.

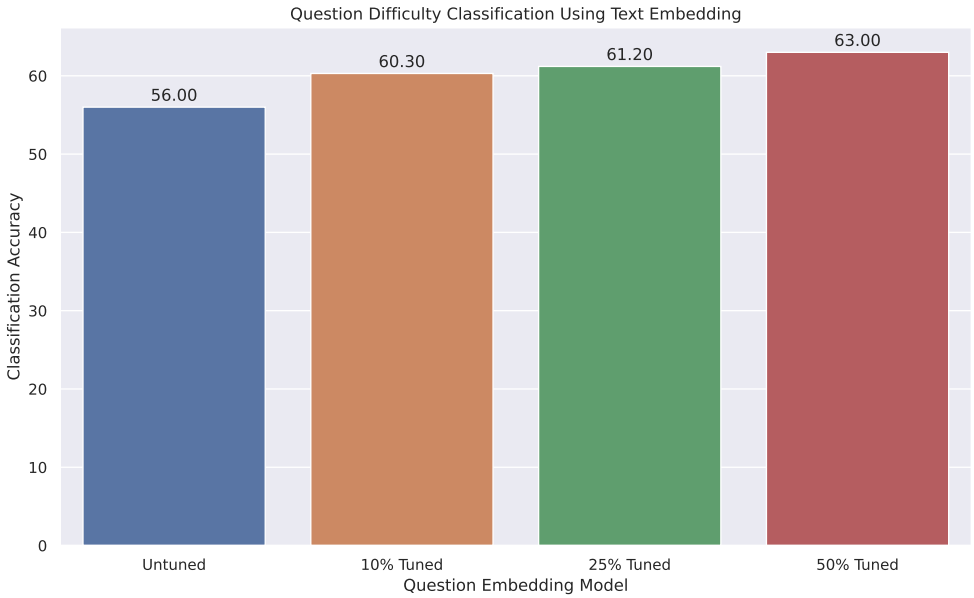


Figure 7.17: A bar chart comparing classification accuracy for different fine-tuned question text embedding variants on classifying question difficulty label task.

Table 7.3: Summary for clustering performance results for fine-tuned variants of question text embedding using Inertia and Homogeneity metrics.

Embedding Method	Inertia	Homogeneity
Ground Truth Embedding	203.8	1.0
10%-tuned Embedding Variant	1870.5	0.71
25%-tuned Embedding Variant	1845.2	0.75
50%-tuned Embedding Variant	1830.6	0.81

classification accuracy with more ground truth data added during the fine-tuning procedure and a significant enhancement over the untuned best performer model from **R1** evaluation.

7.4 Summary

In this chapter, we proposed a new dataset for knowledge tracing, named DBE-KT22. The dataset was collected from real-world exercise answering activities in an undergraduate-level course taught at the Australian National University within the period from 2019 to 2021. The collected data covers a wide range of aspects relating to the knowledge tracing modeling including questions meta-data with text, KCs meta-data with text, relationships among questions and KCs, relationships among KCs and KCs, student feedback on observed difficulty and confidence in answers, and time taken for answering. This can facilitate more machine learning tasks to be formulated based on our dataset such as answer prediction, question text-aware em-

bedding learning, graph representation learning on relationships between questions and KCs, question difficulty prediction, and cognitive analysis based on student feedback. We described the workflow of collecting the dataset and thoroughly investigated its characteristics. Moreover, we performed an experimental evaluation of different methods for learning effective text-aware question embeddings from pre-trained language models. Finally, we made our dataset publicly available through the Australian Data Archive (ADA) platform with an unrestricted access license for easier utilization by researchers in the field of knowledge tracing.

Chapter 8

Conclusion and Future WorK

In this thesis we addressed the knowledge tracing problem across the following four research objectives: 1) modeling long-short term dependencies across questions in an exercise answering sequence, 2) considering the effect of a student's forgetting behavior while estimating their knowledge states, as one of the important psychological aspects that may impact knowledge growth, 3) highlighting the potential of knowledge tracing models in evolving knowledge-augmented data teaching strategies, and 4) addressing limitations in existing knowledge tracing datasets. To achieve these objectives, we proposed three novel models and a new dataset for knowledge-tracing: Sequence Key-Value Memory Network (SKVMN) in Chapter 4, Deep Graph Memory Network (DGMN) in Chapter 5, knowledge Augmented Data Teaching (KADT) in Chapter 6, and Database Exercises dataset (DBE-KT22) in Chapter 7. In addition, we presented a comprehensive literature review for knowledge tracing in Chapter 3.

In this chapter, we conclude our thesis in Section 8.1. Next, we explore future research directions for the KT field, highlighting the opportunities and limitations of existing KT methods in Section 8.2. Finally, we discuss KT application areas in Section 8.3.

8.1 Conclusion

The aim of this thesis was to answer the fundamental question of "Can a machine effectively trace the dynamics of a student's knowledge states?", which is related to the challenging problem called *Knowledge Tracing* (KT). The KT problem can be generally represented as predicting the likelihood of correctly answering a question conditioned on a student's past exercise answering history (i.e., question-answer pairs). We need to predict the probability of correctly answering a new question. Besides the past exercise answering activities, there are additional sources of useful information, including relationships between questions, between learning concepts, and between questions and learning concepts. These relationships are usually represented by a graph that includes nodes for questions and learning concepts with edges linking related nodes.

There are many challenges in addressing the KT problem, including 1) modeling long-short term temporal dynamics in an exercise answering sequence, 2) counting for the student's forgetting behavior that impacts their knowledge states, 3) finding effective ways to consider dependency relationships between questions and learning concepts in a task, and 4) finding effective representations for multidimensional knowledge states that span over multiple learning

concepts in a task. Our research agenda includes four main research questions: **RQ1**) How to effectively represent and track a student's knowledge states given past exercise sequences? **RQ2**) How to represent and consider a student's forgetting behavior for robust knowledge tracing? **RQ3**) Can teaching strategies be learned via knowledge tracing? and **RQ4**) What are the current limitations in KT datasets? How to address them?

In Chapter 4, we proposed a novel deep memory-augmented neural network model named *Sequential Key-Value Memory Networks* (SKVMN) that answers **RQ1** by learning to represent a student's knowledge states as a multidimensional memory matrix over multiple learning concepts, each with its dedicated memory slot. The SKVMN tackled modeling the long-short term dependency challenge by proposing a novel hop technique to attention memory reads and updates on groups of relevant questions over time.

In Chapter 5, we answered **RQ2** by proposing a novel model called *Deep Graph Memory Networks* (DGMN) that counted for relationships between learning concepts using a dynamic graph representation and considered the impact of a student's forgetting behavior using a novel gating mechanism.

In Chapter 6, we addressed **RQ3** by proposing formulating a teaching scenario between a teacher agent and a student model that interacts in a reinforcement learning paradigm to simulate a learning material teaching scenario. Then, we proposed a novel teaching strategy learning framework called *Knowledge-Augmented Data Teaching* (KADT) for learning data teaching strategies in supervised machine learning tasks by firstly tracing a student model's knowledge, then using knowledge representations as states for a reinforcement learning data teaching agent that is rewarded by positive changes in the learning progress of a student model.

In Chapter 7, we respond to **RQ4** by investigating the limitations in existing KT datasets and proposing a new KT dataset called *DBE-KT22* that is collected from real student exercise answering activities in an undergraduate course for databases taught at the Australian National University (ANU). The new dataset overcomes many of the identified limitations and facilitates new ways of addressing the KT problem.

8.2 Future Research Directions

Below, we highlight several possible research directions to extend our work.

8.2.1 Multimodal and Informative Representation Learning

The choice of data representation directly impacts the performance of a machine learning model [13]. KT models tend to learn embedding representations for questions and learning concepts from an abstract format such as one-hot encoding. However, some data in the description of a question, such as images and mathematical equations that may lead to more informative embedding representations, is overlooked, either by the proposed models or by the available datasets.

The Exercise-aware Knowledge Tracing (EKT) approach proposed by Liu et al. [101] is a recent attempt to learn richer embedding representations, considering textual context and relationships between questions. However, despite the previous efforts, the representations of

multimodal and domain-specific data, such as mathematical equations and code snippets, remain mostly unexplored in the literature, resulting in low-informative representation learning for KT models. Combining signals from multiple feature spaces may enable better representation learning and mitigating noise in data [8].

There is a need for a new range of benchmark datasets for KT which can provide contextual information rather than only encoded data with ids. Datasets spanning various knowledge domains (e.g., second language learning courses) and from different cultures and educational levels are also needed, as the performance of KT models can be affected when being applied to other demographic and educational contexts. This leads us to the next research opportunity, Self-Supervised Learning in Knowledge Tracing. Another interesting direction for knowledge tracing datasets is to consider open-ended questions instead of commonly assumed binary-answered questions in the KT literature. Such question types would pose a challenge to existing KT models as there is no obvious way of how to quantify the validity of a student's answer. Liu et al. [100] proposed an approach involving two steps, including 1) quantifying the answer-matching to a modal answer using an NLP-based model and 2) tracing the student's knowledge state using a KT model.

8.2.2 Self-supervised Learning in Knowledge Tracing

Although supervised learning has led to advances in different areas, it still has a major drawback: the need for large, high-quality labeled data for training. Self-supervised learning (SSL) [200, 116], on the other hand, has proved to be effective in several areas (e.g., natural language processing [37] and computer vision [47]) learning from unlabeled data. SSL often adopts similarity ranking loss functions (e.g., contrastive loss [181]) in a process called *pre-training* or *pretext task* [116] to generate labels automatically. Such pre-trained models can thus be transferred to a downstream task to train in a supervised learning manner with a limited amount of labeled data. Therefore, further studies on the KT field can be made, for example, creating pre-trained models (e.g., using existing pre-trained language and computer vision models) to generate informative representations for KT and investigating how it can mitigate limited training of students' activities in cases of a cold-start scenario or skewed participation data, e.g., when only a small number of students contribute to most activities.

8.2.3 Interactive Knowledge Tracing

Most KT models adopt a passive approach of observing question answering response history to estimate students' knowledge states; however, interactive methods driven by question answering response behavior are still unexplored. Interactive methods are particularly useful in cold start scenarios where an interactive approach can reveal students' knowledge states by directly asking questions about different learning concepts. Thus, another potential future work is to develop optimized question sampling policies to enhance the performance of KT models in, but not limited to, cold start scenarios. Reinforcement Learning (RL) [165] approaches are a possible natural choice given their maximal rewarding scheme.

Last but not least, considering that Knowledge Tracking involves human knowledge and learning, transparency in the internal logic and the results obtained by KT models would ben-

efit educational stakeholders and processes. This leads to investigating *eXplainable Artificial Intelligence* (XAI) approaches, as seen in other research fields such as [72, 70]. Potential research avenues include the development of techniques and methods to understand and explain the prediction process in KT models; and how algorithmic decisions impact learning processes, course design, instructor performance, the quality of learning materials, and student engagement. Promising research to explain deep learning models has been carried out using knowledge distillation [59] to understand and explain predictions in other models.

8.3 Knowledge Tracing Applications

This section explores possible application areas that can benefit from KT models. We broadly divide the KT application areas into the following four categories: (i) Educational Recommender Systems; (ii) Learning Provision and Quality Assurance; (iii) Student Engagement via Interactive Learning; and (iv) Machine Teaching.

8.3.1 Educational Recommender Systems

An application area of KT that comes directly to mind is online education systems where the main objective is to provide an effective learning experience for their students. Tracing the knowledge state of a student would facilitate tailoring students' learning experience according to their capabilities and skills. This can be achieved through recommending learning materials (e.g., lectures, labs, and/or exercises) based on the learned knowledge state of a student. Thus, the aim of such online education systems is twofold: (1) to estimate the knowledge state of a student using a KT model; and (2) to recommend learning materials conditioned on the knowledge state using a recommendation model [17]. Below are some examples of applications that have been recently studied. A graph-based recommendation method has been proposed by Chanaa and Faddouli [22] to aid an educational instructor in segmenting students into groups based on their knowledge states and recommend the most appropriate kinds of exercises for each group. More specifically, the instructor first selects a specific learning concept, and then, the model constructs a dynamic knowledge graph based on historical practice information that includes student knowledge vectors as nodes and uses edges to represent their mastery level similarities around the selected learning concept. Such a graph is clustered into node groups, and for each group, a shared embedding is constructed using GNNs to get the final recommendations. Cai et al. [19] followed an interactive recommendation approach in which a reinforcement learning recommender agent selects learning materials to recommend based on a reward signal calculated from the progress in a knowledge state estimated by a KT model. Huang et al. [68] proposed an interactive educational video recommender model that follows a multi-objective rewards setup. The authors designed three reward functions to reflect three main aspects of online education systems, including a reviewing reward function for recommending videos about learning concepts that a student did not perform well previously, a smoothing reward function for recommending videos with a gradual difficulty to understand, and an engagement reward function for recommending videos about learning concepts a student started to master recently. The recommender agent followed a reinforcement

learning design with a state estimated by a DKT variant model, an action for the video id to recommend, and a combined reward by using a weighted sum of the three reward functions.

8.3.2 Learning Provision and Quality Assurance

Another potential application area is to provision a learning curriculum (e.g., an ordered list of topics to study) for a specific subject based on the knowledge state of a student. One direction [31, 152] in this application area follows a semi-automated approach in which an instructor would conduct simulations using a KT model trained on the historical exercise records of students to identify a suitable curriculum of learning materials for a course to maximize the knowledge gain of students. Another direction [129] uses KT models to assess the effectiveness of a course structure in achieving its targeted objectives by assessing the impact of each module (i.e., a collection of learning materials) on the knowledge growth of students. Recently, deep KT models have been adopted in this direction to provide quality assurance for the course design [103]. The authors used a DKT model [137] to trace the progress in the knowledge state of a student after taking a specific course module, and then, an actor-critic reinforcement learning agent [55] considers the knowledge state across different course modules and their predefined relationships (i.e., prerequisites) and takes an action to select the next module to work on for the student to maximize their knowledge gain in achieving course objectives. Following this approach, the structure of a course could adapt dynamically to a student's needs and skills instead of having one fixed structure that does not fit all students.

8.3.3 Student Engagement via Interactive Learning

Interactive education aims at making the learning process more exciting and engaging by delivering knowledge components into a gaming shell. Cognitive studies [62] showed that students can easily gain new knowledge components and be able to spend more time learning if the learning materials were delivered in an engaging manner, such as gaming. This is due to the nature of the human mind, which was mainly designed for learning by real-world practices that usually come as an interactive experience (i.e., state, action, and rewards). Thus, an educational game can provide similar experiences that better align with our natural learning capabilities than the conventional ways (e.g., textbooks and lectures). Long and Aleven [107] evaluated the effect of educational games in comparison to conventional online tutoring systems through a study that involved two groups of students: one group was learning on an educational game for math equation solving and the other group was learning using a non-interactive system that presents math concepts in a conventional manner through demonstrative examples and exercises. The study found that the group using the educational game was more engaged in continuing the learning process in comparison to the other group. Another study [4] focused on the effect of mobile educational games on the learning progress of elementary school kids. The authors divided the student into two groups: one used mobile educational games to review and practice math concepts taught at school, and another group practiced the math concepts using conventional text exercises. The study concluded that students with access to mobile educational games were performing better in retaining math concepts in comparison to the other group. These findings demonstrate the great potential of educational games as a promising

application area for KT models.

Central to the effectiveness of an educational game is to assess the knowledge progress of a player and adjust the gaming experience accordingly. This might include adjusting the difficulty of challenges, opening new part(s) in a game, or the competency of a computer opponent. Kantharaju et al. [75] proposed a KT model that could detect when a player attempts a specific skill in an educational game and quantify their knowledge state across different states so that the game experience could be focused on challenging skills for each player. Cui et al. [32] used the BKT [179] model to trace the knowledge state of fifth-grade elementary school students in Canada during a science gaming assessment. The authors were not only able to effectively predict the final score of a student based on their partial observations from the game assessment but also identify pitfalls in the game design and assumptions that tend not to work as expected by the designer in terms of assessing dedicated skills. Hooshyar et al. [65] proposed a method that tackles the problem of predicting a player's skill proficiency in education games over multiple tasks with overlapping demonstrations between skills. The authors investigated deep knowledge tracing using different sequence models, including LSTMs, RNNs, and CNNs, while decomposing temporal dynamics using a moving fixed-size time window. Their results showed that the CNN variant outperformed others with a classification accuracy of 85% on average. Recently, several studies have shown that psycho-physiological signals could provide vital cues on the current engagement state of a student, especially in virtual learning sessions [15, 175]. These studies use recent advancements in computer vision to detect facial expressions and body poses with a web camera attached to a student's computer for estimating an engagement score. We note that such signals could work as an effective auxiliary source of information for knowledge tracing models besides the question-answering history to better estimate the knowledge state of a student.

8.3.4 Machine Teaching

Going beyond the conventional assumption of a human student in a KT setting opens the door to a wide range of application areas. Virtual students, such as intelligent agents which adopt a reinforcement learning setup or machine learning models, can be treated in a similar way as real-life students who are in need of learning a set of skills in different machine learning tasks. For example, this can be a deep neural network model that needs to master the skill of classifying different class labels (e.g., cats, dogs, furniture, etc.) in an image classification task or a reinforcement learning agent aiming at mastering different skills in an Atari game. Curriculum learning (CL) [14] aims at learning a curriculum of tasks to enable a student agent to master a set of skills. A CL policy would imply a statistical distribution of learning tasks that gradually drive the student agent toward convergence. Another relevant paradigm is machine teaching (MT) [203], which aims at minimizing the teaching cost represented by the size of training samples drawn from training data in a machine learning scenario. In MT, there are two models being included: the teacher model and the student model. The former targets to sample training data for the latter to learn an optimal parameter set θ^* that minimizes the loss function in the task. Learning to teach (L2T) [41, 192] targeted customizing the learning process for a student agent/model through optimizing three main aspects, including training data sampling, neural architecture design, and loss function design. In L2T, a teacher agent

follows a reinforcement learning approach to optimize a teaching policy that handles one or more of the previously mentioned aspects. It can be observed that a shared characteristic across these different attempts to enhance the conventional machine learning procedure is the need to trace the knowledge state of a student model. Thus, there is a significant potential for KT models to contribute to this application area by tracing the knowledge state of a student model during the training procedure. The output of the KT model would form the input/state of a teacher model that aims at customizing the training procedure to speed up the student model's convergence.

Bibliography

1. Abdelrahman, G. and Wang, Q., 2019. Knowledge tracing with sequential key-value memory networks. In *Proceedings of the 42nd International Conference on Research and Development in Information Retrieval, SIGIR,Paris,France*, 175–184. ACM. (cited on pages 2, 3, 24, 35, 44, 48, 52, 94, 98, 105, and 107)

2. Abdelrahman, G. and Wang, Q., 2022. Deep graph memory networks for forgetting-robust knowledge tracing. *IEEE Transactions on Knowledge and Data Engineering,TKDE*, (2022). (cited on pages 4, 24, 42, 44, 52, and 113)

3. Abdelrahman, G.; Wang, Q.; and Nunes, B. P., 2022. Knowledge tracing: A survey. *ACM Computing Surveys*, 55, 11 (2022). (cited on page 23)

4. Al Khateeb, M. A., 2019. Effect of mobile gaming on mathematical achievement among 4th graders. *International Journal of Emerging Technologies in Learning*, 14, 7 (2019). (cited on page 141)

5. Anderson, J. R.; Boyle, C. F.; Corbett, A. T.; and Lewis, M. W., 1990. Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 42, 1 (1990), 7–49. (cited on pages 1, 2, and 3)

6. Andrich, D., 1981. Book review: Probabilistic models for some intelligence and attainment tests (expanded edition: Georg rasch chicago: The university of chicago press, 1980, 199 pp., 15hardcover, 7 paperback. *Applied Psychological Measurement*, 5, 4 (1981), 545–550. (cited on page 29)

7. Bahdanau, D.; Cho, K.; and Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA*. (cited on pages xxi, 44, 83, and 101)

8. Baltrusaitis, T.; Ahuja, C.; and Morency, L.-P., 2019. Multimodal machine learning: A survey and taxonomy. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41, 2 (2019), 423–443. (cited on page 139)

9. Barton, M. A. and Lord, F. M., 1981. An upper asymptote for the three-parameter logistic item-response model. *ETS Research Report Series*, 1981 (1981), i–8. (cited on page 29)

10. Bellman, R., 1957. A markovian decision process. *Journal of mathematics and mechanics*, 6 (1957), 679–684. (cited on page 100)

11. Bellman, R. and Kalaba, R., 1957. Dynamic programming and statistical communication theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43 (1957), 749. (cited on page 100)

12. Bengio, Y., 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade: Second Edition*, 437–478. Springer. (cited on page 64)

13. Bengio, Y.; Courville, A.; and Vincent, P., 2013. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35, 8 (2013), 1798–1828. (cited on page 138)

14. Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J., 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML, Montreal, Quebec, Canada*, vol. 382 of *ACM International Conference Proceeding Series*, 41–48. Association for Computing Machinery. (cited on pages 93 and 142)

15. Bhardwaj, P.; Gupta, P. K.; Panwar, H.; Siddiqui, M. K.; Morales-Menéndez, R.; and Bhaik, A., 2021. Application of deep learning on student engagement in e-learning environments. *Computers & Electrical Engineering*, 93 (2021), 107277. (cited on page 142)

16. Birnbaum, A., 1969. Statistical theory for logistic mental test models with a prior distribution of ability. *Journal of Mathematical Psychology*, 6, 2 (1969), 258–276. (cited on page 29)

17. Bobadilla, J.; Ortega, F.; Hernando, A.; and Gutiérrez, A., 2013. Recommender systems survey. *Knowledge-Based Systems*, 46 (2013), 109–132. (cited on page 140)

18. Bottou, L., 2012. Stochastic gradient descent tricks. *Neural Networks: Tricks of the Trade: Second Edition*, (2012), 421–436. (cited on page 63)

19. Cai, D.; Zhang, Y.; and Dai, B., 2019. Learning path recommendation based on knowledge tracing model and reinforcement learning. In *2019 IEEE 5th International Conference on Computer and Communications ICCC*, 1881–1885. (cited on page 140)

20. Cen, H.; Koedinger, K. R.; and Junker, B., 2006. Learning factors analysis - A general method for cognitive model evaluation and improvement. In *Intelligent Tutoring Systems, 8th International Conference, ITS, Jhongli, Taiwan*, vol. 4053 of *Lecture Notes in Computer Science*, 164–175. Springer. (cited on pages 3, 30, and 44)

21. Cen, H.; Koedinger, K. R.; and Junker, B., 2008. Comparing two IRT models for conjunctive skills. In *Intelligent Tutoring Systems, 9th International Conference, ITS,Montreal, Canada*, vol. 5091, 796–798. Springer. (cited on pages 3, 30, 44, and 52)

22. Chanaa, A. and El Faddouli, N.-E., 2020. Predicting learners need for recommendation using dynamic graph-based knowledge tracing. In *Artificial Intelligence in Education, AIED, Ifrane, Morocco*, vol. 12164 of *Lecture Notes in Computer Science*, 49–53. Springer. (cited on page 140)

23. Chang, H.; Hsu, H.; and Chen, K., 2015. Modeling exercise relationships in e-learning: A unified approach. In *Proceedings of the 8th International Conference on Educational Data Mining, EDM*, 532–535. (cited on pages 48 and 117)

24. Chen, Y.; Liu, Q.; Huang, Z.; Wu, L.; Chen, E.; Wu, R.; Su, Y.; and Hu, G., 2017. Tracking knowledge proficiency of students with educational priors. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM*, 989–998. (cited on pages 4, 10, 41, and 44)

25. Cheng, S.; Liu, Q.; Chen, E.; Zhang, K.; Huang, Z.; Yin, Y.; Huang, X.; and Su, Y., 2022. Adaptkt: A domain adaptable method for knowledge tracing. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, WSDM,Tempe, AZ, USA*, 123–131. ACM. (cited on pages 40 and 44)

26. Choi, Y.; Lee, Y.; Cho, J.; Baek, J.; Kim, B.; Cha, Y.; Shin, D.; Bae, C.; and Heo, J., 2020. Towards an appropriate query, key, and value computation for knowledge tracing. In *L@S'20: Seventh ACM Conference on Learning @ Scale, Virtual Event, USA*, 341–344. (cited on pages 3, 24, 36, 37, 44, and 52)

27. Choi, Y.; Lee, Y.; Shin, D.; Cho, J.; Park, S.; Lee, S.; Baek, J.; Bae, C.; Kim, B.; and Heo, J., 2020. Ednet: A large-scale hierarchical dataset in education. In *Artificial Intelligence in Education - 21st International Conference, AIED*, vol. 12164 of *Lecture Notes in Computer Science*, 69–73. Springer. (cited on pages 50 and 117)

28. Chun, B. A. and Heo, H. J., 2018. The effect of flipped learning on academic performance as an innovative method for overcoming ebbinghaus' forgetting curve. In *Proceedings of the 6th International Conference on Information and Education Technology,ICEMT*, 56–60. (cited on pages xviii and 10)

29. Chung, J.; Gülçehre, Ç.; Cho, K.; and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555 (2014). http://arxiv.org/abs/1412.3555. (cited on page 13)

30. Corbett, A., 2000. Cognitive mastery learning in the act programming tutor. In *Adaptive User Interfaces. AAAI SS-00-01. Retrieved from https://aaai.org/Library/Symposia/Spring/ss00-01. php*. (cited on page 25)

31. Corbett, A. T. and Anderson, J. R., 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction, UMUAI*, 4, 4 (1994), 253–278. (cited on pages 4, 10, 23, 25, 27, 28, 32, 44, 52, 55, 64, and 141)

32. Cui, Y.; Chu, M.-W.; and Chen, F., 2019. Analyzing student process data in game-based assessments with bayesian knowledge tracing and dynamic bayesian networks. *Journal of Educational Data Mining*, 11, 1 (2019), 80–100. (cited on page 142)

33. de Baker, R. S. J.; Corbett, A. T.; and Aleven, V., 2008. More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In *Intelligent Tutoring Systems, 9th International Conference, ITS , Montreal,*

Canada, vol. 5091 of *Lecture Notes in Computer Science*, 406–415. Springer. (cited on pages xxi, 3, 25, 32, and 44)

34. de Baker, R. S. J.; Pardos, Z. A.; Gowda, S. M.; Nooraei, B. B.; and Heffernan, N. T., 2011. Ensembling predictions of student knowledge within intelligent tutoring systems. In *User Modeling, Adaption and Personalization - 19th International Conference, UMAP 2011, Girona, Spain*, 13–24. (cited on page 1)

35. Defferrard, M.; Bresson, X.; and Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems,NeurIPS, Barcelona, Spain*, 3837–3845. (cited on page 20)

36. Desmarais, M. C. and d Baker, R. S., 2012. A review of recent advances in learner and skill modeling in intelligent learning environments. *User Modeling and User-Adapted Interaction, UMUAI*, 22 (2012), 9–38. (cited on page 10)

37. Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K., 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, Minneapolis, MN, USA*, 4171–4186. Association for Computational Linguistics. (cited on pages 130 and 139)

38. Duvenaud, D.; Maclaurin, D.; Aguilera-Iparraguirre, J.; Gómez-Bombarelli, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P., 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems, NeurIPS, Montreal, Quebec, Canada*, 2224–2232. (cited on pages 20 and 80)

39. Ebbinghaus, H., 2013. Memory: A contribution to experimental psychology. *Annals of neurosciences*, 20, 4 (2013), 155. (cited on pages 2, 10, 41, and 74)

40. Embretson, S. E. and Reise, S. P., 2013. *Item response theory*. Psychology Press. (cited on pages 29 and 43)

41. Fan, Y.; Tian, F.; Qin, T.; Li, X.; and Liu, T., 2018. Learning to teach. In *6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, Conference Track Proceedings*. OpenReview.net. (cited on pages 94, 102, 106, 107, 114, and 142)

42. Feng, M.; Heffernan, N.; and Koedinger, K., 2009. Addressing the assessment challenge with an online system that tutors as it assesses. *User modeling and user-adapted interaction,UMUAI*, 19, 3 (2009), 243–266. (cited on pages 45 and 117)

43. Foster, G. C.; Min, H.; and Zickar, M. J., 2017. Review of item response theory practices in organizational research: Lessons learned and paths forward. *Organizational Research Methods*, 20, 3 (2017), 465–486. (cited on page 48)

44. Gan, W.; Sun, Y.; Peng, X.; and Sun, Y., 2020. Modeling learner's dynamic knowledge construction procedure and cognitive item difficulty for knowledge tracing. *Applied Intelligence*, 50, 11 (2020), 3894–3912. (cited on pages 3, 31, 44, and 52)

45. Ghosh, A.; Heffernan, N.; and Lan, A. S., 2020. Context-aware attentive knowledge tracing. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA*, 2330–2339. (cited on pages 3, 24, 36, 44, 51, 52, 83, 85, and 90)

46. Glorot, X. and Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics, AISTATS, Chia Laguna Resort, Sardinia, Italy*, vol. 9 of *JMLR Proceedings*, 249–256. JMLR.org. (cited on pages 63 and 84)

47. Goyal, P.; Caron, M.; Lefaudeux, B.; Xu, M.; Wang, P.; Pai, V.; Singh, M.; Liptchinsky, V.; Misra, I.; Joulin, A.; and Bojanowski, P., 2021. Self-supervised pretraining of visual features in the wild. *CoRR*, abs/2103.01988 (2021). https://arxiv.org/abs/2103.01988. (cited on page 139)

48. Graves, A., 2013. Generating Sequences With Recurrent Neural Networks. *arXiv e-prints*, (Aug 2013), arXiv:1308.0850. (cited on pages 11 and 32)

49. Graves, A.; Bellemare, M. G.; Menick, J.; Munos, R.; and Kavukcuoglu, K., 2017. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, vol. 70, 1311–1320. (cited on page 93)

50. Graves, A.; Jaitly, N.; and Mohamed, A., 2013. Hybrid speech recognition with deep bidirectional LSTM. In *IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic*, 273–278. (cited on page 11)

51. Graves, A.; Mohamed, A.; and Hinton, G. E., 2013. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing,ICASSP, Vancouver, BC, Canada*, 6645–6649. IEEE. (cited on pages 11, 32, and 55)

52. Graves, A.; Wayne, G.; and Danihelka, I., 2014. Neural turing machines. *CoRR*, abs/1410.5401 (2014). http://arxiv.org/abs/1410.5401. (cited on pages 3, 17, 34, 79, and 99)

53. Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwinska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J. P.; Badia, A. P.; Hermann, K. M.; Zwols, Y.; Ostrovski, G.; Cain, A.; King, H.; Summerfield, C.; Blunsom, P.; Kavukcuoglu, K.; and Hassabis, D., 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538, 7626 (2016), 471–476. (cited on pages 6, 15, 55, and 60)

54. Green, B. F., 1951. A general solution for the latent class model of latent structure analysis. *Psychometrika*, 16, 2 (1951), 151–166. (cited on pages 29, 44, and 52)

55. Grondman, I.; Busoniu, L.; Lopes, G. A. D.; and Babuska, R., 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions*

on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42, 6 (2012), 1291–1307. (cited on page 141)

56. Hamilton, W. L.; Ying, Z.; and Leskovec, J., 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems, NeurIPS, Long Beach, CA, USA*, 1024–1034. (cited on page 20)

57. Hamp-Lyons, L. and Mathias, S. P., 1994. Examining expert judgments of task difficulty on essay tests. *Journal of Second Language Writing*, 3, 1 (1994), 49–68. (cited on page 118)

58. Hasan, S. A.; Zhao, S.; Datla, V. V.; Liu, J.; Lee, K.; Qadir, A.; Prakash, A.; and Farri, O., 2016. Clinical question answering using key-value memory networks and knowledge graph. In *Proceedings of The Twenty-Fifth Text REtrieval Conference, TREC , Gaithersburg, Maryland, USA*, vol. 500-321 of *NIST Special Publication*. National Institute of Standards and Technology (NIST). (cited on page 15)

59. Haselhoff, A.; Kronenberger, J.; Kuppers, F.; and Schneider, J., 2021. Towards black-box explainability with gaussian discriminant knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops*, 21–28. (cited on page 140)

60. Hassan, A. and Mahmood, A., 2017. Deep learning approach for sentiment analysis of short texts. In *3rd international conference on control, automation and robotics, (ICCAR)*, 705–710. IEEE. (cited on page 107)

61. Hawkes, A. G., 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58, 1 (1971), 83–90. (cited on page 41)

62. Hawlitschek, A. and Joeckel, S., 2017. Increasing the effectiveness of digital educational games: The effects of a learning instruction on students' learning, motivation and cognitive load. *Computers in Human Behavior*, 72 (2017), 79–86. (cited on page 141)

63. He, K.; Zhang, X.; Ren, S.; and Sun, J., 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 770–778. (cited on page 107)

64. Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural Computation*, 9, 8 (1997), 1735–1780. (cited on pages 12, 33, and 62)

65. Hooshyar, D.; Huang, Y.-M.; and Yang, Y., 2022. Gamedkt: Deep knowledge tracing in educational games. *Expert Systems with Applications*, 196 (2022), 116670. (cited on page 142)

66. Huang, Y.; González-Brenes, J. P.; and Brusilovsky, P., 2014. General features in knowledge tracing to model multiple subskills, temporal item response theory, and expert knowledge. In *Proceedings of the 7th International Conference on Educational Data Mining, EDM, London, UK*, 84–91. (cited on page 32)

67. Huang, Z.; Liu, Q.; Chen, Y.; Wu, L.; Xiao, K.; Chen, E.; Ma, H.; and Hu, G., 2020. Learning or forgetting? a dynamic approach for tracking the knowledge proficiency of students. *ACM Trans. Inf. Syst.*, 38, 2 (2020), 19:1–19:33. (cited on pages 41 and 44)

68. Huang, Z.; Liu, Q.; Zhai, C.; Yin, Y.; Chen, E.; Gao, W.; and Hu, G., 2019. Exploring multi-objective exercise recommendations in online education systems. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management,CIKM, Beijing, China*, 1261–1270. ACM. (cited on page 140)

69. Hubert, L. and Arabie, P., 1985. Comparing partitions. *Journal of classification*, 2, 1 (1985). (cited on page 132)

70. Ignatiev, A., 2020. Towards trustable explainable AI. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, 5154–5158. (cited on page 140)

71. Ilse, M.; Tomczak, J. M.; and Welling, M., 2018. Attention-based deep multiple instance learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML, Stockholmsmässan, Stockholm, Sweden,*, vol. 80 of *Proceedings of Machine Learning Research*, 2132–2141. PMLR. (cited on page 101)

72. Islam, S. R.; Eberle, W.; and Ghafoor, S. K., 2020. Towards quantification of explainability in explainable artificial intelligence methods. In *Proceedings of the Thirty-Third International Florida Artificial Intelligence Research Society Conference,AAAI*, 75–81. (cited on page 140)

73. Jiang, L.; Meng, D.; Yu, S.; Lan, Z.; Shan, S.; and Hauptmann, A. G., 2014. Self-paced learning with diversity. In *Advances in Neural Information Processing Systems, NeurIPS*, 2078–2086. (cited on page 93)

74. Jr., P. I. P. and Anderson, J. R., 2005. Practice and forgetting effects on vocabulary memory: An activation-based model of the spacing effect. *Cognitive Science*, 29, 4 (2005), 559–586. (cited on pages 3, 5, and 40)

75. Kantharaju, P.; Alderfer, K. B.; Zhu, J.; Char, B.; Smith, B. K.; and Ontañón, S., 2018. Tracing player knowledge in a parallel programming educational game. In *Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE, Edmonton, Canada*, 173–179. (cited on page 142)

76. Käser, T.; Klingler, S.; Schwing, A. G.; and Gross, M. H., 2017. Dynamic bayesian networks for student modeling. *IEEE Trans. Learn. Technol.*, 10, 4 (2017), 450–462. (cited on pages xxi, 28, 32, and 44)

77. Khajah, M.; Huang, Y.; González-Brenes, J. P.; Mozer, M.; and Brusilovsky, P., 2014. Integrating knowledge tracing and item response theory: A tale of two frameworks. In *Workshop Proceedings of the 22nd Conference on User Modeling, Adaptation, and Personalization, UMAP, Aalborg, Denmark*, vol. 1181 of *CEUR Workshop Proceedings*. CEUR-WS.org. (cited on page 32)

78. Khajah, M.; Wing, R.; Lindsey, R. V.; and Mozer, M., 2014. Integrating latent-factor and knowledge-tracing models to predict individual differences in learning. In *Proceedings of the 7th International Conference on Educational Data Mining, EDM 2014, London, UK*, 99–106. (cited on pages 2, 27, 28, 32, 41, and 63)

79. Kingma, D. P. and Ba, J. L., 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA*. (cited on page 64)

80. Kipf, T. N. and Welling, M., 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR, Toulon, France, Conference Track Proceedings*. OpenReview.net. (cited on pages 3, 20, 21, 76, and 80)

81. Kiranyaz, S.; Ince, T.; Abdeljaber, O.; Avci, O.; and Gabbouj, M., 2019. 1-d convolutional neural networks for signal processing applications. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8360–8364. (cited on pages xxi, 37, and 44)

82. Klir, G. J. and Yuan, B., 1996. Fuzzy sets and fuzzy logic: theory and applications. *Possibility Theory versus Probab. Theory*, 32, 2 (1996), 207–208. (cited on page 61)

83. Koedinger, K. R.; Baker, R. S.; Cunningham, K.; Skogsholm, A.; Leber, B.; and Stamper, J., 2010. A data repository for the edm community: The pslc datashop. *Handbook of educational data mining*, 43 (2010), 43–56. (cited on pages 47 and 117)

84. Krizhevsky, A.; Hinton, G.; et al., 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer. (cited on page 106)

85. Krizhevsky, A.; Sutskever, I.; and Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems, NeurIPS, Lake Tahoe, Nevada, United States*, 1106–1114. (cited on page 20)

86. Kumar, A.; Irsoy, O.; Ondruska, P.; Iyyer, M.; Bradbury, J.; Gulrajani, I.; Zhong, V.; Paulus, R.; and Socher, R., 2016. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of the 33nd International Conference on Machine Learning, ICML , New York City, NY, USA*, vol. 48 of *JMLR Workshop and Conference Proceedings*, 1378–1387. JMLR.org. (cited on page 15)

87. Kumar, M. P.; Packer, B.; and Koller, D., 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems , NeurIPS, Vancouver, British Columbia, Canada*, 1189–1197. Curran Associates, Inc. (cited on pages 93, 106, and 107)

88. Lanchantin, J.; Singh, R.; and Qi, Y., 2017. Memory matching networks for genomic sequence classification. In *5th International Conference on Learning Representations, ICLR , Toulon, France, Workshop Track Proceedings*. OpenReview.net. (cited on page 16)

89. Lauritzen, S. L.; Dawid, A. P.; Larsen, B. N.; and Leimer, H., 1990. Independence properties of directed markov fields. *Networks*, 20, 5 (1990), 491–505. (cited on page 11)

90. Le, H.; Tran, T.; and Venkatesh, S., 2018. Dual control memory augmented neural networks for treatment recommendations. In *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD , Melbourne, VIC, Australia, Proceedings, Part III*, vol. 10939 of *Lecture Notes in Computer Science*, 273–284. Springer. (cited on page 15)

91. LeCun, Y. and Bengio, Y., 1998. *Convolutional Networks for Images, Speech, and Time Series*, 255–258. MIT Press, Cambridge, MA, USA. (cited on page 20)

92. LeCun, Y.; Bengio, Y.; and Hinton, G., 2015. Deep learning. *nature*, 521, 7553 (2015), 436. (cited on pages 3, 32, and 55)

93. Lee, J. I. and Brunskill, E., 2012. The impact on individualizing student models on necessary practice opportunities. In *Proceedings of the 5th International Conference on Educational Data Mining, EDM, Chania, Greece*, 118–125. (cited on pages 27 and 28)

94. Lee, Y. J. and Grauman, K., 2011. Learning the easy things first: Self-paced visual category discovery. In *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 1721–1728. (cited on page 93)

95. Levin, E., 1990. A recurrent neural network: Limitations and training. *Neural Networks*, 3, 6 (1990), 641–650. (cited on page 12)

96. Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D., 2016. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico*. (cited on page 103)

97. Ling, C. X.; Huang, J.; and Zhang, H., 2003. Auc: A statistically consistent and more discriminating measure than accuracy. In *Proceedings of the 18th International Joint Conference on Artificial IntelligenceIJCAI, Acapulco, Mexico*, 519–526. Morgan Kaufmann. (cited on pages 64 and 84)

98. Lipton, Z. C.; Berkowitz, J.; and Elkan, C., 2015. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019 (2015). http://arxiv.org/abs/1506.00019. (cited on pages 3 and 33)

99. Liu, J. and Zhu, X., 2016. The teaching dimension of linear learners. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, vol. 48, 117–126. (cited on page 93)

100. Liu, N.; Wang, Z.; Baraniuk, R. G.; and Lan, A. S., 2022. Open-ended knowledge tracing for computer science education. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP, Abu Dhabi, United Arab Emirates*, 3849–3862. Association for Computational Linguistics. (cited on page 139)

101. Liu, Q.; Huang, Z.; Yin, Y.; Chen, E.; Xiong, H.; Su, Y.; and Hu, G., 2021. EKT: exercise-aware knowledge tracing for student performance prediction. *IEEE Trans. Knowl. Data Eng.*, 33, 1 (2021), 100–115. (cited on pages 4, 40, 44, and 138)

102. Liu, Q.; Shen, S.; Huang, Z.; Chen, E.; and Zheng, Y., 2021. A survey of knowledge tracing. *arXiv preprint arXiv:2105.15106*, (2021). (cited on page 2)

103. Liu, Q.; Tong, S.; Liu, C.; Zhao, H.; Chen, E.; Ma, H.; and Wang, S., 2019. Exploiting cognitive structure for adaptive learning. In *Proceedings of the 25th ACM International Conference on Knowledge Discovery & Data Mining,SIGKDD, Anchorage, AK, USA*, 627–635. Association for Computing Machinery. (cited on page 141)

104. Liu, W.; Dai, B.; Humayun, A.; Tay, C.; Yu, C.; Smith, L. B.; Rehg, J. M.; and Song, L., 2017. Iterative machine teaching. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, vol. 70, 2149–2158. (cited on page 93)

105. Liu, Y.; Yang, Y.; Chen, X.; Shen, J.; Zhang, H.; and Yu, Y., 2021. Improving knowledge tracing via pre-training question embeddings. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, 1577–1583. ijcai.org. (cited on pages 38, 39, and 44)

106. Long, T.; Qin, J.; Shen, J.; Zhang, W.; Xia, W.; Tang, R.; He, X.; and Yu, Y., 2022. Improving knowledge tracing with collaborative information. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, WSDM, Tempe, AZ, USA*, 599–607. ACM. (cited on pages 36, 38, and 44)

107. Long, Y. and Aleven, V., 2017. Educational game and intelligent tutoring system: A classroom study and comparative design analysis. *ACM Trans. Comput.-Hum. Interact.*, 24, 3 (2017), 20:1–20:27. (cited on pages 1 and 141)

108. Lord, F.; Novick, M.; and Birnbaum, A., 1968. *Statistical theories of mental test scores.* Addison-Wesley. (cited on page 29)

109. Lord, F. M., 1951. A theory of test scores and their relation to the trait measured. *ETS Research Bulletin Series*, 1951, 1 (1951), i–126. (cited on page 29)

110. Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C., 2011. Learning word vectors for sentiment analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, HLT, Portland, Oregon, USA*, 142–150. The Association for Computer Linguistics. (cited on page 105)

111. McCue, T., 2018. E learning climbing to $325 billion by 2025 UF canvas absorb schoology moodle. *Forbes*, (2018). https://www.forbes.com/sites/tjmccue/2018/07/31/e-learning-climbing-to-325-billion-by-2025-uf-canvas-absorb-schoology-moodle/#688469483b39. (cited on page 4)

112. Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J., 2013. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR,Scottsdale, Arizona, USA, Workshop Track Proceedings*. (cited on page 40)

113. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems, NeurIPS, Lake Tahoe, Nevada, United States*, 3111–3119. (cited on page 14)

114. Miller, A. H.; Fisch, A.; Dodge, J.; Karimi, A.; Bordes, A.; and Weston, J., 2016. Key-value memory networks for directly reading documents. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP, Austin, Texas, USA*, 1400–1409. The Association for Computational Linguistics. (cited on pages xviii, xxi, 3, 19, 34, and 44)

115. Minn, S.; Yu, Y.; Desmarais, M. C.; Zhu, F.; and Vie, J., 2018. Deep knowledge tracing and dynamic student classification for knowledge tracing. In *International Conference on Data Mining, ICDM, Singapore*, 1182–1187. IEEE Computer Society. (cited on pages 34, 44, 51, and 52)

116. Misra, I. and Maaten, L. v. d., 2020. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, Seattle, WA, USA*, 6706–6716. (cited on page 139)

117. Montavon, G.; Orr, G. B.; and Müller, K. (Eds.), 2012. *Neural Networks: Tricks of the Trade - Second Edition*, vol. 7700 of *Lecture Notes in Computer Science*. Springer. (cited on page 14)

118. Montero, S.; Arora, A.; Kelly, S.; Milne, B.; and Mozer, M., 2018. Does deep knowledge tracing model interactions among skills? In *Proceedings of the 11th International Conference on Educational Data Mining, EDM , Buffalo, NY, USA*. International Educational Data Mining Society (IEDMS). (cited on page 23)

119. Murray, R. C.; Ritter, S.; Nixon, T.; Schwiebert, R.; Hausmann, R. G. M.; Towle, B.; Fancsali, S. E.; and Vuong, A., 2013. Revealing the learning in learning curves. In *Artificial Intelligence in Education - 16th International Conference, AIED 2013, Memphis, TN, USA, July 9-13, 2013. Proceedings*, vol. 7926 of *Lecture Notes in Computer Science*, 473–482. Springer. (cited on pages 3, 4, 10, and 41)

120. Murre, J. M. J. and Dros, J., 2015. Replication and analysis of Ebbinghaus' forgetting curve. *PLOS ONE*, 10, 7 (2015), 1–23. (cited on pages 10 and 41)

121. Nagatani, K.; Zhang, Q.; Sato, M.; Chen, Y.; Chen, F.; and Ohkuma, T., 2019. Augmenting knowledge tracing by considering forgetting behavior. In *International Conference on World Wide Web, WWW, San Francisco, CA, USA*, 3101–3107. ACM. (cited on pages 4, 41, 44, 52, 73, 83, and 85)

122. Nakagawa, H.; Iwasawa, Y.; and Matsuo, Y., 2019. Graph-based knowledge tracing: Modeling student proficiency using graph neural network. In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI (Thessaloniki, Greece, 2019), 156–163. Association for Computing Machinery, New York, NY, USA. (cited on pages 3, 24, 38, 44, 45, 52, 83, 85, and 86)

123. Novak, J. D. and Cañas, A. J., 2008. The theory underlying concept maps and how to construct and use them. *Florida Institute for Human and Machine Cognition*, (2008). (cited on page 75)

124. Ontañón, S., 2020. An overview of distance and similarity functions for structured data. *Artificial Intelligence Review*, (2020). (cited on page 84)

125. Oudeyer, P.; Kaplan, F.; and Hafner, V. V., 2007. Intrinsic motivation systems for autonomous mental development. *IEEE Trans. Evol. Comput.*, 11, 2 (2007), 265–286. (cited on page 102)

126. Pandey, S. and Karypis, G., 2019. A self attentive model for knowledge tracing. In *Proceedings of the 12th International Conference on Educational Data Mining, EDM , Montréal, Canada*. International Educational Data Mining Society (IEDMS). (cited on pages 3, 24, 36, 44, 51, 52, 83, 85, and 90)

127. Pandey, S. and Srivastava, J., 2020. Rkt: Relation-aware self-attention for knowledge tracing. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management,CIKM,Virtual Event, Ireland*, 1205–1214. ACM. (cited on pages 3, 24, 36, 37, 40, 44, 48, and 52)

128. Pang, B. and Lee, L., 2008. Opinion mining and sentiment analysis. *Foundations and Trends® in information retrieval*, 2, 1–2 (2008), 1–135. (cited on page 11)

129. Pardos, Z.; Bergner, Y.; Seaton, D.; and Pritchard, D., 2013. Adapting bayesian knowledge tracing to a massive open online course in edx. In *Proceedings of the 6th International Conference on Educational Data Mining, EDM, Memphis, Tennessee, USA*, 137–144. International Educational Data Mining Society. (cited on page 141)

130. Pardos, Z. A.; Baker, R. S.; San Pedro, M. O.; Gowda, S. M.; and Gowda, S. M., 2014. Affective states and state tests: Investigating how affect and engagement during the school year predict end-of-year learning outcomes. *Journal of Learning Analytics*, 1, 1 (2014), 107–128. (cited on page 45)

131. Pardos, Z. A. and Heffernan, N. T., 2010. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *Proceedings of the 18th International Conference on User Modeling, Adaptation, and Personalization, UMAP,Big Island, HI, USA*, vol. 6075 of *Lecture Notes in Computer Science*, 255–266. Springer. (cited on pages 27, 28, and 32)

132. Pardos, Z. A. and Heffernan, N. T., 2011. KT-IDEM: introducing item difficulty to the knowledge tracing model. In *User Modeling, Adaption and Personalization - 19th*

International Conference,UMAP, Girona, Spain, vol. 6787 of *Lecture Notes in Computer Science*, 243–254. Springer. (cited on page 32)

133. Pascanu, R.; Mikolov, T.; and Bengio, Y., 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML, Atlanta, GA, USA*, vol. 28 of *JMLR Workshop and Conference Proceedings*, 1310–1318. JMLR.org. (cited on pages 12 and 64)

134. Pavlik, P. I.; Cen, H.; and Koedinger, K. R., 2009. Performance factors analysis - A new alternative to knowledge tracing. In *Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling, AIED, Brighton, UK*, vol. 200 of *Frontiers in Artificial Intelligence and Applications*, 531–538. IOS Press. (cited on pages 3, 30, 44, and 52)

135. Pelánek, R., 2015. Modeling students' memory for application in adaptive educational systems. In *Proceedings of the 8th International Conference on Educational Data Mining, EDM, Madrid, Spain*, 480–483. (cited on pages 40 and 41)

136. Pennington, J.; Socher, R.; and Manning, C. D., 2014. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 1532–1543. ACL. (cited on page 14)

137. Piech, C.; Bassen, J.; Huang, J.; Ganguli, S.; Sahami, M.; Guibas, L. J.; and Sohl-Dickstein, J., 2015. Deep knowledge tracing. In *Advances in Neural Information Processing Systems, NeurIPS, Montreal, Quebec, Canada*, 505–513. (cited on pages 3, 10, 23, 33, 34, 39, 41, 43, 44, 48, 52, 55, 63, 64, 83, 94, 107, 111, 117, and 141)

138. Psotka, J.; Massey, L. D.; and Mutter, S. A., 1988. *Intelligent tutoring systems: Lessons learned*. Psychology Press. (cited on pages 1 and 9)

139. Qiu, Y.; Qi, Y.; Lu, H.; Pardos, Z. A.; and Heffernan, N. T., 2011. Does time matter? modeling the effect of time with bayesian knowledge tracing. In *Proceedings of the 4th International Conference on Educational Data Mining, EDM, Eindhoven, The Netherlands*, 139–148. (cited on page 41)

140. Radvansky, G. A.; Doolen, A. C.; Pettijohn, K. A.; and Ritchey, M., 2022. A new look at memory retention and forgetting. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 48, 11 (2022), 1698. (cited on page 10)

141. Rasch, G., 1993. *Probabilistic models for some intelligence and attainment tests.* ERIC. (cited on pages 29 and 37)

142. Rathore, A. S. and Arjaria, S. K., 2020. Intelligent tutoring system. In *Utilizing educational data mining techniques for improved learning: emerging research and opportunities*, 121–144. IGI Global. (cited on page 9)

143. Rendle, S. and Schmidt-Thieme, L., 2008. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys, Lausanne, Switzerland*, 251–258. (cited on page 107)

144. Robertson, S. E. and Zaragoza, H., 2009. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3, 4 (2009), 333–389. (cited on page 38)

145. Rosenberg, A. and Hirschberg, J., 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL, Prague, Czech Republic*, 410–420. ACL. (cited on page 132)

146. Sanders, D. and Welk, D. S., 2005. Strategies to scaffold student learning: Applying vygotsky's zone of proximal development. *Nurse educator*, 30, 5 (2005), 203–207. (cited on page 93)

147. Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; and Lillicrap, T., 2016. One-shot learning with memory-augmented neural networks. *CoRR*, abs/1605.06065 (2016). http://arxiv.org/abs/1605.06065. (cited on page 6)

148. Santoro, A.; Bartunov, S.; Botvinick, M. M.; Wierstra, D.; and Lillicrap, T. P., 2016. Meta-learning with memory-augmented neural networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML, New York City, NY, USA*. (cited on pages 6, 15, 55, 82, and 84)

149. Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G., 2009. The graph neural network model. *IEEE Transactions on Neural Networks*, 20, 1 (2009), 61–80. (cited on pages xxi, 3, 20, 38, and 44)

150. Schmidhuber, J., 2010. Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Trans. Auton. Ment. Dev.*, 2, 3 (2010), 230–247. (cited on page 102)

151. Schmidhuber, J., 2015. Deep learning in neural networks: An overview. *Neural Networks*, 61 (2015), 85 – 117. (cited on pages 11 and 32)

152. Schodde, T.; Bergmann, K.; and Kopp, S., 2017. Adaptive robot language tutoring based on bayesian knowledge tracing and predictive decision-making. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction,HRI, Vienna, Austria,*, 128–136. Association for Computing Machinery. (cited on page 141)

153. Sense, F.; Behrens, F.; Meijer, R. R.; and van Rijn, H., 2016. An individual's rate of forgetting is stable over time but differs across materials. *Topics in cognitive science*, 8, 1 (2016), 305–321. (cited on page 40)

154. Settles, B. and Meeder, B., 2016. A trainable spaced repetition model for language learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL, Berlin, Germany*, 1848–1858. The Association for Computer Linguistics. (cited on page 41)

155. Shen, S.; Liu, Q.; Chen, E.; Huang, Z.; Huang, W.; Yin, Y.; Su, Y.; and Wang, S., 2021. Learning process-consistent knowledge tracing. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Singapore*, 1452–1460. (cited on pages 42 and 44)

156. Shen, S.; Liu, Q.; Chen, E.; Wu, H.; Huang, Z.; Zhao, W.; Su, Y.; Ma, H.; and Wang, S., 2020. Convolutional knowledge tracing: Modeling individualization in student learning process. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, China*, 1857–1860. ACM. (cited on pages 36, 37, and 44)

157. Shin, D.; Shim, Y.; Yu, H.; Lee, S.; Kim, B.; and Choi, Y., 2021. SAINT+: integrating temporal features for ednet correctness prediction. In *LAK'21: 11th International Learning Analytics and Knowledge Conference, Irvine, CA, USA*, 490–496. ACM. (cited on pages 3, 36, 37, 44, 52, 83, and 85)

158. Simard, P. Y.; Amershi, S.; Chickering, D. M.; Pelton, A. E.; Ghorashi, S.; Meek, C.; Ramos, G. A.; Suh, J.; Verwey, J.; Wang, M.; and Wernsing, J., 2017. Machine teaching: A new paradigm for building machine learning systems. *arXiv CoRR*, (2017). (cited on page 93)

159. Spitkovsky, V. I.; Alshawi, H.; and Jurafsky, D., 2010. From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics,HLT*, 751–759. (cited on page 93)

160. Stamper, J.; Niculescu-Mizil, A.; Ritter, S.; Gordon, G.; and Koedinger, K., 2010. Algebra i 2008–2009. challenge data set from kdd cup 2010 educational data mining challenge. *Retrieved April*, 25 (2010), 2015. (cited on pages 48 and 117)

161. Su, Y.; Liu, Q.; Liu, Q.; Huang, Z.; Yin, Y.; Chen, E.; Ding, C. H. Q.; Wei, S.; and Hu, G., 2018. Exercise-enhanced sequential modeling for student performance prediction. In *Proceedings of the Thirty-Second Conference on Artificial Intelligence, New Orleans, Louisiana, USA*, 2435–2443. (cited on pages 4, 39, 40, and 44)

162. Sukhbaatar, S.; Szlam, A.; Weston, J.; and Fergus, R., 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems, NeurIPS, Montreal, Quebec, Canada*, 2440–2448. (cited on pages 15 and 16)

163. Sutskever, I.; Vinyals, O.; and Le, Q. V., 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems,NeurIPS, Montreal, Quebec, Canada*, 3104–3112. (cited on pages 11, 32, and 55)

164. Sutton, R. S. and Barto, A. G., 1998. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press. (cited on pages 93, 94, 102, and 103)

165. Sutton, R. S. and Barto, A. G., 2018. *Reinforcement learning: An introduction*. MIT press. (cited on page 139)

166. Sweeney, M.; Lester, J.; Rangwala, H.; and Johri, A., 2016. Next-term student performance prediction: A recommender systems approach. In *Proceedings of the 9th International Conference on Educational Data Mining, EDM, Raleigh, North Carolina, USA*, 7. International Educational Data Mining Society (IEDMS). (cited on pages xxi, 31, and 44)

167. Szabados, T., 2010. An elementary introduction to the wiener process and stochastic integrals. *arXiv preprint arXiv:1008.1510*, (2010). (cited on page 30)

168. Tang, S.; Peterson, J. C.; and Pardos, Z. A., 2016. Modelling student behavior using granular large scale action data from a mooc. *CoRR*, abs/1608.04789 (2016). http://arxiv.org/abs/1608.04789. (cited on page 1)

169. Thai-Nghe, N.; Drumond, L.; Horváth, T.; and Schmidt-Thieme, L., 2012. Using factorization machines for student modeling. In *Workshop and Poster Proceedings of the 20th Conference on User Modeling, Adaptation, and Personalization, UMAP, Montreal, Canada*. (cited on pages xxi, 31, and 44)

170. Thorndike, R. L., 1953. Who belongs in the family. In *Psychometrika*. (cited on page 132)

171. Tolstikhin, I. O.; Sriperumbudur, B. K.; and Schölkopf, B., 2016. Minimax estimation of maximum mean discrepancy with radial kernels. In *Advances in Neural Information Processing Systems,NeurIPS, Barcelona, Spain*, 1930–1938. (cited on page 40)

172. Tong, H.; Wang, Z.; Zhou, Y.; Tong, S.; Han, W.; and Liu, Q., 2022. Introducing problem schema with hierarchical exercise graph for knowledge tracing. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain*, 405–415. ACM. (cited on page 40)

173. Tong, S.; Liu, Q.; Huang, W.; Huang, Z.; Chen, E.; Liu, C.; Ma, H.; and Wang, S., 2020. Structure-based knowledge tracing: An influence propagation view. In *20th IEEE International Conference on Data Mining, ICDM,Sorrento, Italy*, 541–550. (cited on pages 3, 24, 38, 39, 44, 48, and 52)

174. Van der Maaten, L. and Hinton, G., 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9, 11 (2008). (cited on page 132)

175. Vanneste, P.; Oramas, J.; Verelst, T.; Tuytelaars, T.; Raes, A.; Depaepe, F.; and Van den Noortgate, W., 2021. Computer vision and human behaviour, emotion and cognition detection: A use case on student engagement. *Mathematics*, 9, 3 (2021). (cited on page 142)

176. Vardi, M. Y., 2012. Will MOOCs destroy academia? *Communications of the ACM*, 55 (2012), 5–5. (cited on page 1)

177. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I., 2017. Attention is all you need. In *Advances in Neural Information*

Processing Systems, NeurIPS, Long Beach, CA, USA, 6000–6010. (cited on pages xxi, 3, 14, 36, 37, 38, 44, 83, and 101)

178. Vie, J. and Kashima, H., 2019. Knowledge tracing machines: Factorization machines for knowledge tracing. In *The Thirty-Third Conference on Artificial Intelligence, IAAA, Honolulu, Hawaii,USA*, 750–757. (cited on pages 3, 10, 31, 32, 44, and 52)

179. Villano, M., 1992. Probabilistic student models: Bayesian belief networks and knowledge space theory. In *Intelligent Tutoring Systems, Second International Conference, ITS, Montréal, Canada, Proceedings*, vol. 608 of *Lecture Notes in Computer Science*, 491–498. Springer. (cited on pages xxi, 1, 3, 25, 43, 44, and 142)

180. Wang, C.; Ma, W.; Zhang, M.; Lv, C.; Wan, F.; Lin, H.; Tang, T.; Liu, Y.; and Ma, S., 2021. Temporal cross-effects in knowledge tracing. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining, WSDM , Israel*, 517–525. (cited on pages 41, 44, 51, 52, 83, and 85)

181. Wang, F. and Liu, H., 2021. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2495–2504. Computer Vision Foundation / IEEE. (cited on page 139)

182. Wang, F.; Liu, Q.; Chen, E.; Huang, Z.; Chen, Y.; Yin, Y.; Huang, Z.; and Wang, S., 2020. Neural cognitive diagnosis for intelligent education systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 6153–6161. AAAI Press. (cited on page 30)

183. Wang, L.; Sy, A.; Liu, L.; and Piech, C., 2017. Learning to represent student knowledge on programming exercises using deep learning. In *Proceedings of the 10th International Conference on Educational Data Mining, EDM 2, Wuhan, Hubei, China*. International Educational Data Mining Society (IEDMS). (cited on page 32)

184. Wang, Z.; Lamb, A.; Saveliev, E.; Cameron, P.; Zaykov, J.; Hernandez-Lobato, J. M.; Turner, R. E.; Baraniuk, R. G.; Barton, C.; Jones, S. P.; et al., 2021. Results and insights from diagnostic questions: The neurips 2020 education challenge. In *NeurIPS Competition and Demonstration Track*, 191–205. PMLR. (cited on page 50)

185. Wang, Z.; Lamb, A.; Saveliev, E.; Cameron, P.; Zaykov, Y.; Hernández-Lobato, J. M.; Turner, R. E.; Baraniuk, R. G.; Barton, C.; Jones, S. P.; et al., 2020. Instructions and guide for diagnostic questions: The neurips 2020 education challenge. *arXiv preprint arXiv:2007.12061*, (2020). (cited on page 50)

186. Wass, R.; Harland, T.; and Mercer, A., 2011. Scaffolding critical thinking in the zone of proximal development. *Higher Education Research & Development*, 30, 3 (2011), 317–328. (cited on page 93)

187. Watkins, C. J. and Dayan, P., 1992. Q-learning. *Machine learning*, 8, 3 (1992), 279–292. (cited on page 30)

188. Weston, J., 2016. Dialog-based language learning. In *Advances in Neural Information Processing Systems, NeurIPS, Barcelona, Spain*, 829–837. (cited on page 16)

189. Weston, J.; Chopra, S.; and Bordes, A., 2014. Memory networks. *arXiv preprint arXiv:1410.3916*, (2014). (cited on page 16)

190. Wilson, K. H.; Karklin, Y.; Han, B.; and Ekanadham, C., 2016. Back to the basics: Bayesian extensions of IRT outperform neural networks for proficiency estimation. In *Proceedings of the 9th International Conference on Educational Data Mining, EDM, Raleigh, North Carolina, USA*, 539–544. (cited on pages 30, 44, 46, 50, and 52)

191. Wright, R. E., 1995. Logistic regression. *Circulation*, (1995). (cited on pages xxi, 3, and 44)

192. Wu, L.; Tian, F.; Xia, Y.; Fan, Y.; Qin, T.; Lai, J.; and Liu, T., 2018. Learning to teach with dynamic loss functions. In *Advances in Neural Information Processing Systems , NeurIPS, Montréal, Canada*, 6467–6478. (cited on pages 94 and 142)

193. Xiong, X.; Zhao, S.; Inwegen, E. V.; and Beck, J., 2016. Going deeper with deep knowledge tracing. In *Proceedings of the 9th International Conference on Educational Data Mining, EDM, Raleigh, North Carolina, USA*, 545–550. International Educational Data Mining Society (IEDMS). (cited on pages 23, 34, and 45)

194. Xu, H.; Farajtabar, M.; and Zha, H., 2016. Learning granger causality for hawkes processes. In *Proceedings of the 33nd International Conference on Machine Learning, ICML, New York City, NY, USA*, vol. 48 of *JMLR Workshop and Conference Proceedings*, 1717–1726. (cited on page 83)

195. Yang, Y.; Shen, J.; Qu, Y.; Liu, Y.; Wang, K.; Zhu, Y.; Zhang, W.; and Yu, Y., 2020. GIKT: A graph-based interaction model for knowledge tracing. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD , Ghent, Belgium, Proceedings, Part I*, vol. 12457 of *Lecture Notes in Computer Science*, 299–315. Springer. (cited on pages 3, 24, 38, 39, 44, 51, and 52)

196. Yang, Z.; Chen, Y.; Hong, M.; and Wang, Z., 2019. Provably global convergence of actor-critic: A case for linear quadratic regulator with ergodic cost. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems , NeurIPS , Vancouver, BC, Canada*, 8351–8363. (cited on page 103)

197. Yeung, C. and Yeung, D., 2018. Addressing two problems in deep knowledge tracing via prediction-consistent regularization. In *Proceedings of the Fifth Annual Conference on Learning at Scale, L@S, London, UK*, 5:1–5:10. ACM. (cited on pages 34, 44, 48, 51, and 52)

198. Yin, Y.; Liu, Q.; Huang, Z.; Chen, E.; Tong, W.; Wang, S.; and Su, Y., 2019. Quesnet: A unified representation for heterogeneous test questions. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD , Anchorage, AK,USA*, 1328–1336. (cited on pages 4 and 40)

199. Yudelson, M. V.; Koedinger, K. R.; and Gordon, G. J., 2013. Individualized bayesian knowledge tracing models. In *Artificial Intelligence in Education - 16th International Conference,AIED, Memphis, TN, USA*, vol. 7926 of *Lecture Notes in Computer Science*, 171–180. Springer. (cited on pages 27 and 28)

200. Zhai, X.; Oliver, A.; Kolesnikov, A.; and Beyer, L., 2019. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision, ICCV, Seoul, Korea (South)*, 1476–1485. (cited on page 139)

201. Zhang, J.; Shi, X.; King, I.; and Yeung, D., 2017. Dynamic key-value memory networks for knowledge tracing. In *Proceedings of the 26th International Conference on World Wide Web, WWW, Perth, Australia*, 765–774. ACM. (cited on pages 2, 3, 24, 34, 44, 46, 47, 51, 52, 55, 57, 60, 63, 64, 66, 83, 85, 90, 94, 98, and 105)

202. Zhou, Y.; Liu, Q.; Wu, J.; Wang, F.; Huang, Z.; Tong, W.; Xiong, H.; Chen, E.; and Ma, J., 2021. Modeling context-aware features for cognitive diagnosis in student learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Virtual Event, Singapore*, 2420–2428. ACM. (cited on page 30)

203. Zhu, X., 2015. Machine teaching: An inverse problem to machine learning and an approach toward optimal education. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29, 1 (2015). (cited on pages 93 and 142)

204. Zhuang, Y.; Liu, Q.; Huang, Z.; Li, Z.; Shen, S.; and Ma, H., 2022. Fully adaptive framework: Neural computerized adaptive testing for online education. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 4734–4742. AAAI Press. (cited on page 30)