FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Knowlege tracing techniques, their explainability and applications
Student:	Bc. Eliška Svobodová
Supervisor:	doc. Ing. Pavel Kordík, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2024/2025

Instructions

The primary objective of this thesis is to investigate the application of knowledge tracing within a novel personalized educational platform (informatika.gg). The research will involve a comprehensive examination of knowledge tracing techniques, exploring the diverse inputs and outputs of existing methods with a particular emphasis on their explainability and visualization capabilities. A curated list of datasets suitable for knowledge tracing will be compiled, along with a detailed description of their characteristics and the benefits of incorporating various student and learning process metadata. The thesis will then proceed with the implementation of several knowledge tracing models, showcasing their potential inputs, outputs, visualizations, and approaches to enhancing explainability. The models will be trained and evaluated on datasets selected from the list of datasets compiled in the previous step. Finally, the advantages and disadvantages of the implemented approaches will be thoroughly discussed. The goal of this thesis is to offer several prototypes for the platform showcasing possible applications of knowledge tracing in personalized education.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 12 December 2023 in Prague.

Master's thesis

KNOWLEGE TRACING TECHNIQUES, THEIR EXPLAINABILITY AND APPLICATIONS

Bc. Eliška Svobodová

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: doc. Ing. Pavel Kordík, Ph.D.
May 9, 2024

Citation of this thesis: Svobodová Eliška. *Knowlege tracing techniques, their explainability and applications.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of Tables

I want to thank my supervisor doc. Ing. Pavel Kordík, Ph.D. for his invaluable guidance and support throughout the development of this thesis. I am also deeply grateful to my family for their unwavering support and encouragement.

Declaration

In Prague on May 9, 2024

Abstract

Swift technological advancement and the ever-evolving needs of society lead to the growing importance of personalized online education. A core challenge of personalization is knowledge tracing - a machine-learning task of predicting a student's performance while interacting with learning materials.

This thesis explores knowledge tracing and proposes a categorization of existing methods according to their primary mechanics. It identifies the main representatives within each category and describes their prediction process and visualization techniques. A list of knowledge tracing datasets from literature is compiled featuring their key metrics. Additionally, a new dataset is created from data gathered in a FIT CTU's quiz system Marast. Subsequently, datasets utilized for the experimentaion part of this thesis are described in greater detail.

A representative of each category of deep knowledge tracing techniques is implemented, compared with others, and extensively analyzed. Furthermore, KT models were modified to process the text of questions. The analysis showed the promise of knowledge tracing for the next exercise personalized difficulty prediction. Notably, models leveraging attention mechanism and memory-augmentation provide interpretable visualizations further explaining the predicted difficulty.

Keywords education, personalization, knowledge tracing

Abstrakt

Technologický pokrok spojený s vývojem potřeb společnosti vyžaduje personalizované online vzdělávání. Jeho nedílnou součástí je modelování znalostí studenta - knowledge tracing, což je úloha strojového učení pro předpovídání studentovy úspěšnosti na testovacích úlohách z jeho předchozích interakcí se studijními materiály.

Tato diplomová práce popisuje výzkum úlohy knowledge tracing a navrhuje způsob jeho kategorizace dle primárních mechanik modelů. Proces předpovědi a techniky vizualizace jsou popsány pro zástupce každé kategorie. Je uveden seznam datasetů spolu s jejich hlavními vlastnostmi a metrikami. Práce také zahrnuje vytvoření nového datsetu z dat z fakultního kvízového systému Marast. Datasety později využité pro experimenty jsou popsány ve větším detailu.

Práce dále pokračuje implementací knowledge tracing modelů, jejich srovnáním s ostatními a zevrubnou analýzou. Byl implementován model z každé kategorie a nově upraveny modely pro zpracování textu otázek. Analýza ukázala potenciál úlohy knowledge tracing pro předpovídání personalizované obtížnosti otázek. Modely s pozornostními mechanismy (attention mechanisms) a explicitní pamětí vedly k nejintuitivnějším vizualizacím, které dále vysvětlují předpovězenou obtížnost otázek.

Klíčová slova vzdělávání, personalizace, modelování znalostí

Abbreviations

KT	Knowledge Tracing
KC	Knowledge Concept
NN	Neural Network
ML	Machine Learning
BKT	Bayesian Knowledge Tracing
FA	Factor Analysis
KTM	Knowledge Tracing Machine
IRT	Item Response Theory
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory
LLM	Large Language Model
GNN	Graph Neural Network
DKT	Deep Knowledge Tracing
DKVMN	Dynamic Key-Value Memory Network
SAKT	Self-Attentive model for Knowledge Tracing
GKT	Graph-based Knowledge Tracing
ACC	Accuracy
ROC	Receiver Operating Characteristic Curve
AUC	Area Under the ROC Curve

Chapter 1

Introduction

Education empowers the advancement of modern society. However, the current educational framework often falls short in adapting to the dynamic demands of society and taking advantage of rapidly advancing technologies. One of the most pressing problems is the lack of personalization. Each student possesses different learning abilities and interests and as a result, tailored curriculum and learning materials promise a better understanding of the topic and more motivation for students for follow-up learning. Unfortunately, the traditional educational system isn't able to accommodate that as teachers are responsible for a whole class of students at once and many classes throughout the week. Because of this one-size-fits-all approach, all students are usually assigned the same study work and are evaluated on a common scale. This demotivates students who are weaker in the subject and subsequently kills their curiosity and drive for further learning. On the other hand, talented students often become bored during the classes and they are not provided with materials advanced enough which is a waste of their potential.

With this in mind, the project informatika.gg aims to create a personalized educational platform, leveraging artificial intelligence (AI) to analyze and improve students' learning journey. The first goal is to develop a mobile application that assists the teacher during lectures and provides tailored support for each child. The app will display learning materials, evaluate students' understanding of the topic, and recommend next steps to the teacher. The student will get personalized explanations or extending pieces of information. The project focuses on improving education of informatics and AI, because nowadays, artificial intelligence promises to revolutionize the way we work, create, and solve everyday problems. Therefore, teaching children how AI works and how to use it safely and efficiently is crucial. However, the platform should be applicable to any subject.

A necessary prerequisite for personalizing an educational system is an insight into a student's knowledge state – determining which concepts the student understands and which need to be further explained. That is the goal of knowledge tracing (KT) - to model a student's inner knowledge state. To turn this intention into an optimizable machine learning (ML) task, researchers defined knowledge tracing as a sequence prediction problem where the KT model predicts a probability of the student answering the next question correctly based on previous interactions with the educational platform. It does that based on its representation of the student's inner knowledge state which can be a set of parameters, a vector, or a more sophisticated structure.

To better grasp the concept of knowledge, we divide the student's knowledge state into the understanding of smaller parts of the topic which we call knowledge concepts (KC) or skills. These parts can have different level of complexity and are usually related to each other as prerequisites or by sharing a similar idea. For example, a KC can be something as fundamental as understanding multiplication, or the skill of solving a polynomial.

This thesis aims to explore the field of knowledge tracing and propose its possible meaningful applications for a novel education platform informatika.gg. In the literature review part, I summarize the current state of research of knowledge tracing and describe the most significant models, techniques, and ideas. I focus on exploring the diversity of the KT models - their variable inputs and outputs, the inner representation of student's knowledge state, and the mechanics connecting the two. Another important feature that is thoroughly studied is the explainability and visualization capability of the KT models. A list of datasets used in KT research was compiled with their key metrics and metadata. I have selected several datasets from this list suitable for further usage and describe these in more detail. Additionally, for use in further research, I provide a description of important characteristics and features of a knowledge tracing dataset.

The second part of this thesis is the implementation of several KT models and their evaluation. Besides the standard prediction accuracy comparison, a real-world utilization of the models is investigated, for example, which visualizations and features for the education platform the model might provide. The results and findings from the experimentation are discussed along with the possible future directions.

Chapter 2
Literature review

Knowledge tracing (KT) is a machine learning task with a rich history and model taxonomy. It aims at predicting a student's future performance based on their learning history. To accurately predict whether the student will answer the next question correctly, the KT model maintains (usually a dynamic) representation of the student's knowledge state, which models a current level of understanding or mastery of a particular subject or domain. This representation can take many forms and is one of the most significant aspects that differentiate the KT models. The representation can encapsulate the entirety of a student's knowledge in one vector or it can be broken down into distinctive units of knowledge called knowledge concepts (KC).

A common assumption is that the question or exercise requires sufficient knowledge of one or more KCs to be solved correctly. In other words, a student who has mastered the required skills has a higher probability of answering the question correctly. The probability is never 100 % as anybody can make a mistake (because of stress, inattention or misunderstanding the assignment). Nevertheless, handling multiple KCs per question proved to be one of the problems the KT researchers had to tackle.

Another obstacle is the sparseness of KT data connected with a cold start problem (i.e. the need to predict correctly even when not enough interactions of a student has been observed yet). A specific student usually interacts with only a handful of exercises from the massive amount of exercises. Additionally, the model of knowledge state is needed as soon as possible and is used to predict the success probability for the student's next question which includes previously unseen exercises. A common solution is to replace the question ID with the required KC ID during modeling because the number of KCs is mostly much smaller than the number of exercises. As a result, the KT model predicts the probability of correct answers on KCs using the records of how successful the student was in those KCs (instead of questions). However, even though the sparseness problem is mitigated, we are losing information and granularity of the interactions.

Definition and Terminology

Knowledge Tracing (KT) is a machine learning task that takes in a sequence of interactions $x_0, \dots x_t$ and predicts the probability that the student answers the next item correctly.

- interaction x_i = a unit of student's history, a tuple of an item, its correctness, and optionally other features $(m_i, a_i, (f_i))$
- item m_i = a unit of KT model input and output, either a question (exercise) q_i, or a knowledge concept (skill) c_i required by the question (depends on the granularity of the task)
- correctness of an answer a_i = binary evaluation of the student's solution of the given item

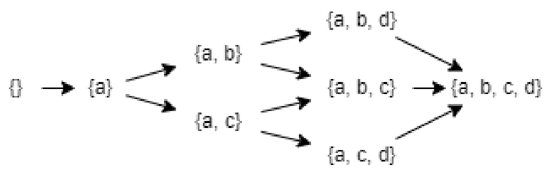


Figure 2.1 Example knowledge space of Bayesian Belief Network BKT

Additionally, the total number of questions in the dataset is denoted as Q, and the total number of knowledge concepts as C. A common assumption is that there are many more questions than KCs ($C << Q$).

Knowledge tracing methods are usually categorized into traditional machine learning methods and deep learning methods. The traditional KT is based on statistics and probability theory featuring Bayesian models and factor analysis, whereas deep learning KT takes advantage of the greater representation ability of various types of neural networks. Although deep learning proves to be much more robust and generally achieves better accuracy, traditional models usually showcase superior explainability.

2.1 Traditional Machine Learning Methods

Traditional methods employ probability theory and statistics to directly model the parameters that influence learning, such as prior knowledge, probability of mastering the KC or probability of a correct guess. This gives them superior explainability over deep learning models because we know exactly what each parameter represents and we can explain its role in the prediction function. There are two primary categories: Bayesian knowledge tracing (BKT) and Item Response Theory (IRT). BKT describes the learning with probabilistic graphical models (explained in further text) and updates them using the Bayes rule. IRT aggregates multiple learning parameters into a one-dimensional latent knowledge state and infers a probability of a correct answer with an item response function.

2.1.1 Bayesian Knowledge Tracing

BKT directly models the learning parameters with a probabilistic graphical model (PGM). PGMs are statistical models that encode the conditional dependencies between random variables into a graph. The kind of PGMs mostly used in BKT are Hidden Markov Models (HMMs), or Bayesian Belief Networks. Student's knowledge state is captured with a set of binary values – one value is used for each KC, where value 1 represents that the student understands the concept, and value 0 suggests that the student has not mastered the KC yet. The values are updated with PGMs based on the observed interactions.

Bayesian Belief Network BKT (BBN-BKT)

One of the first BKT papers [1] used the Bayesian Belief Network to track the student's learning progress. The directed acyclic graph called knowledge space represents student's learning path with each node capturing a combination of mastered KCs (example shown in figure 2.1).

Not all combinations of mastered KCs are feasible because a student cannot master a KC without mastering its preliminaries first. This implies that the KC dependencies are captured in the structure of the BBN. Additionally, each edge represents an acquisition of knowledge of a certain KC meaning the neighboring knowledge states differ only in one mastered KC. In other words, the probability of a student mastering KCs $\{a, b, c\}$ depends on the probability of a student mastering $\{a, b\}$.

Each node is assigned a conditional probability from the training data which can be later updated with the newly observed interactions. From the constructed BBN we can calculate the probability of a student mastering a combination of KCs or a single KC. The KC graph is either given by an expert, or it can be estimated from empirical data (the authors suggested using the increasing item difficulty), or it can be constructed by another model (for example a neural network).

To summarize, the input of the model is a KC and the output is its estimated probability of being mastered at that moment. We can also get a probability of mastering a combination of several KCs. Regarding explainability, we have a probability of mastering each KC in each time step with a curriculum graph specifying possible learning paths.

Standard BKT

Standard BKT, introduced in [2] is based on the Hidden Markov Model with two states for each KC - the KC is either learned or unlearned with transition possible only from learned to unlearned (forgetting the KC is not possible). We have an estimate of the probability of the student being in the learned state for each KC. When observing an interaction, the model updates its estimates according to it.

More precisely, each KC is represented with a model consisting of four parameters:

- Prior knowledge $p(L_0)$ = probability that the student had mastered the KC before starting the exercise sequence
- Transition $p(T)$ = probability of a transition from an unlearned state to a learned state following the exercise
- Guess $p(G)$ = probability a student will guess correctly if the KC is in the unlearned state
- Slip $p(S)$ = probability a student will make a mistake if the KC is in the learned state

The process of inference is the following: first, the initial probability of a student mastering the KC is set to the value of the prior knowledge parameter (equation 2.1). The probability of a student correctly answering a question in time step t can be calculated as shown in equation 2.2 where we utilize the estimated probability of mastering the skill $p(L_t)$ at time step t and the model's parameters (probabilities of slip and guess). Finally, we update the mastery estimation with the observed result. The conditional probability given the observation is calculated using either equation 2.3 (when the answer at time step t was correct) or 2.4 (when the student answered incorrectly). This probability is used in equation 2.5 to get the new estimate.

$$p(L_1) = p(L_0) \tag{2.1}$$
$$p(y^t) = p(L_t) \cdot (1 - p(S)) + (1 - p(L_t)) \cdot p(G) \tag{2.2}$$
$$p(L_{t+1}|obs = correct) = \frac{p(L_t) \cdot (1 - p(S))}{p(L_t) \cdot (1 - p(S)) + (1 - p(L_t)) \cdot p(G)} \tag{2.3}$$
$$p(L_{t+1}|obs = wrong) = \frac{p(L_t) \cdot p(S)}{p(L_t) \cdot p(S) + (1 - p(L_t)) \cdot (1 - p(G))} \tag{2.4}$$
$$p(L_{t+1}) = p(L_{t+1}|obs) + (1 - p(L_{t+1}|obs)) \cdot p(T) \tag{2.5}$$

The input to the standard BKT model is the sequence of the correctness of answers to questions regarding the queried KC. The output is the probability of correctly answering the KC in the next step. Each parameter of the BKT model has a known role and is therefore easily interpretable. However, the model has many limitations. For example, it assumes that each question requires only one skill and does not take into account the dependencies between skills. It also ignores temporal dimension, specifically the fact that students tend to forget concepts after some time.

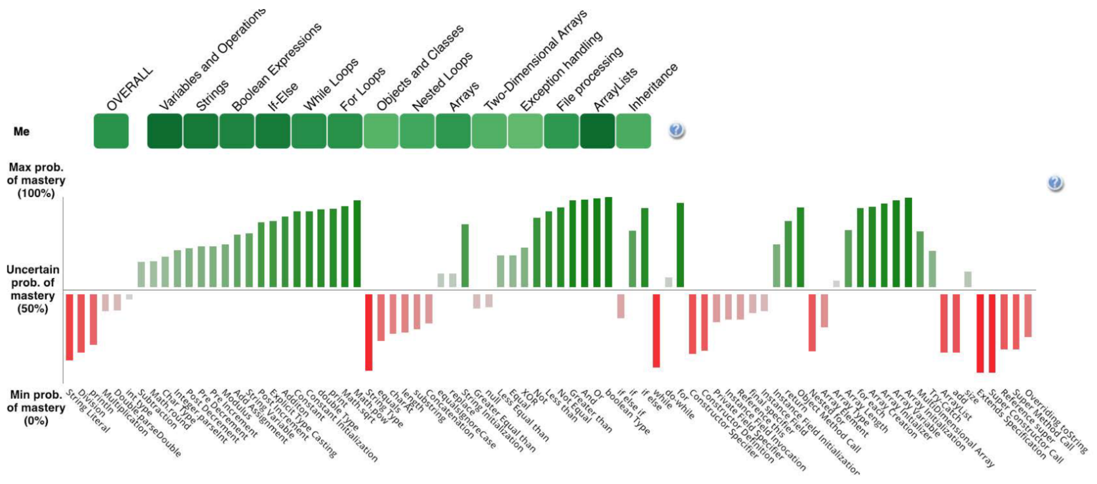


Figure 2.2 Example of visualization of student's knowledge state estimated by BKT from [5]

Weakest Knowledge Tracing model (WKT)

To overcome the limitation of one skill per question, the Weakest Knowledge Tracing extension to BKT was introduced in [3]. It trains a separate BKT model for each skill on an expanded dataset. The correct answer is included in training sequences of all skills required in the question, whereas a record of a wrong answer is appended only to the sequence of the weakest skill required by the question. The assumption is that the skill the student knows the least is responsible for the mistake. Therefore, during prediction on a multi-KCs question, the model calculates the probability of answering each of the KCs correctly and outputs the lowest one. In a real-world scenario, the student can be shown the predicted probabilities of all KCs and be recommended exercises practicing the weakest ones.

Personalized BKT (PBKT)

Another extension to BKT was proposed in [4] which introduces personalization to BKT. The authors conducted a systematic experiment to find out which parameters lead to the best results with personalization. They trained the BKT model with different combinations of personalized and non-personalized prior knowledge parameter $p(L_0)$ and transition probability $p(T)$. The authors discovered that $p(L_0)$ offers only a slight improvement, whereas individualizing $p(T)$ led to significant accuracy improvement. This means that it is advantegous to consider the different learning speeds of the students.

Vizualization from BKT

Even though BKT models were mostly replaced by the deep KT, they are still sometimes used for their superior explainability. For example, [5] uses the BKT model for explaining exercise recommendations. They visualize the level of each skill and the probability of solving the next exercise (which is selected to be a challenge for the student, but solvable with the acquired KCs). An example interface from the paper is shown in figure 2.2.

Corrigible Knowledge Tracing (CKT)

The Corrigible Knowledge Tracing model from [6] extends WKT with the idea that students can learn from their mistakes. In WKT's expansion of the dataset only the weakest skill gets a record of an incorrect answer. In contrast, in CKT when a student answers incorrectly, skills with a mastery level over a threshold get a record of a correctly answered question, and all skills under the threshold are updated with a record of an incorrectly answered question. The intuition is that the stronger skills cannot be blamed for the wrong answer and were still practiced (therefore a positive record), while all weaker skills are a probable cause of the wrong answer (not just the weakest one). Additionally, the authors tried to capture the fact that even when the student knows two KCs, they still needs to understand how to connect them and apply them together. Therefore, they added skill pairs of the most common skills to the skill pool.

To summarize the Bayesian KT, the learning parameters are directly modeled through probabilistic graphical models. Commonly, we train one model per KC with its own set of parameters, which gives us an opportunity to compare the KCs (for example, how difficult to learn they are). Moreover, the parameters can be personalized for each student (even though not all parameters lead to significantly better performance as shown in [4]). Also, BKT models differ in dataset utilization as there are varying strategies regarding sub-dataset selection for each KC (CKT [6], WKT [3]).

2.1.2 Item Response Theory

Item Response Theory (IRT) is a set of mathematical models for finding a relationship between a latent trait and its manifestation. In our case, IRT links the student's inner knowledge state with their performance on a test. The item response function captures the relationship, if four assumptions hold:

1. Unidimensionality = The observed responses are mainly determined by one trait.
2. Monotonicity = The item response function monotonically increases with the student's ability.
3. Local independence = Items are considered conditionally independent given a certain level of ability.
4. Invariance = The model can estimate the item's parameters from any position on the item response curve. We can estimate the item's parameters from any group of students who answered the item.

Classical IRT assigns each student a static mastery level, and each question a difficulty parameter. These parameters are trained to maximize the log posterior probability given the response data. Assuming that the student i has been assigned a mastery level θ_i and each question has been assigned a difficulty β_j. The one-parameter version of IRT, known as the Rash model [7], is

$$p_j(\theta_i) = \frac{1}{1 + e^{-(\theta_i - \beta_j)}}$$

Furthermore, the paper [8] suggested including information about the structure of exercises to the IRT, creating Hierarchical IRT (HIRT). The exercises are organized into groups where exercises forming a specific group are considered to have the same difficulty distribution. Another extension from [8] is Temporal IRT (TIRT) which captures students' knowledge changing over time. The knowledge state is modeled as a stochastic process.

Factor Analysis

Factor Analysis (FA) is a method used to reduce a large number of parameters into fewer latent variables called factors. It was applied in the field of Knowledge Tracing to allow the dependency of a single question on multiple knowledge concepts. It adds the learning rate parameter of the skill into IRT models.

$$p_{ij} = \mathcal{L}(\theta_i + \sum_{k \in K(j)} (\beta_k + \gamma_k T_{ik})) \qquad (2.6) \qquad p_{ij} = \mathcal{L}(\sum_{k \in K(j)} (\beta_k + \gamma_k^S T_{ik}^S + \gamma_k^F T_{ik}^F)) \qquad (2.7)$$

β_k = difficulty of the skill k
γ_k = learning rate of the skill k
γ_k^S = learning rate of the skill k during successful attempt
T_{ik} = number of times student i attempted question with the skill k
T_{ik}^S = number of times student i successfully attempted a question with the skill k
$K(j)$ = set of skills required by the question j
$\mathcal{L}(\cdot)$ = logistic function

Figure 2.3 Comparison of equations for skill-mastery prediction for AFM and PFA.

$$\psi(p(x)) = \mu + \sum_{k=1}^{N} w_k x_k + \sum_{1 \le k < l \le N} x_k x_l \langle \boldsymbol{v_k}, \boldsymbol{v_l} \rangle \qquad (2.8)$$

$p(x)$ = probability of observing a positive outcome (correctly answered question)
μ = global bias
w_k = feature weights
N = length of feature vector x
$\boldsymbol{v_i}$ = embedding of a feature i
$\psi(\cdot)$ = link function

Figure 2.4 Knowledge Tracing Machine

The first variations including factor analysis are the Additive Factor Model (AFM) from [9] and the Parametric Factor Analysis model (PFA) introduced in [10]. AFM focuses on differences between skills - it assumes that some skills are harder to learn than others and there is a different probability that the student knows the skill before learning. The model is personalized only with differing prior knowledge of students and the learning rate is the same for all students. The limitation of this model is that it considers only the number of uses of the specific skill, disregarding whether they were successful or not. PFA overcomes this by splitting the general learning rate into learning rate for successful attempts and learning rate for unsuccessful attempts. Equations in figure 2.3 illustrate the difference.

Knowledge Tracing Machine

Knowledge Tracing Machines (KTM) are extensions of FA and were first proposed in [11] as an application of factorization machines to knowledge tracing. A factorization machine is a model for capturing interactions between features, which is intended for sparse data. To apply them to KT, the authors proposed how to encode the learning process into a vector of features x. They assumed a fixed number of students, questions, and skills. The vector x is made of one-hot encoding of a student, one-hot encoding of a question, binary encoding of which skills are required by the question, a counter denoting how many times was each skill used successfully and another counter for unsuccessful attempts. The KTM equation is shown in figure 2.4. Interestingly, when $\psi(\cdot)$ is a logit function, KTM includes AFM and PFA.

The paper [12] added a cognitive question difficulty and forgetting behavior to KTM. Cog-

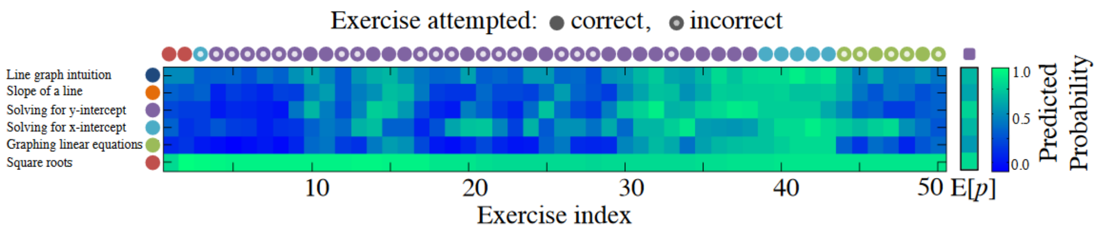


Figure 2.5 Visualization from the first deep KT research article [13]. It shows evolving probabilities of answering the KCs correctly for one randomly selected student.

nitive question difficulty is a personalized difficulty based on the student's past answers and the question's general difficulty (same for all students). They model the learning and forgetting behavior from temporal lags between the last successful usage of a skill.

2.2 Deep Learning Methods

A student's learning process is influenced by many factors such as the form and difficulty level of learning materials, the surrounding environment, and the ability, emotional state and personality of the student themselves. Therefore neural networks, renowned for their capacity to model complex relationships, attracted the attention of KT researchers. Neural networks were first applied to KT in [13] where authors introduced the Deep Knowledge Tracing (DKT) model. Since then, deep KT models dictated the state-of-the-art performance. They utilize modern techniques and approaches for neural networks such as memory augmentation, attention mechanism and graph convolutions. That allows them to consider information that traditional models cannot, especially complex relationships between questions and skills, textual features of learning materials, or advanced patterns in the question sequences. Generally, the price for these performance improvements is lower explainability of the models and results (even though there has been an effort to improve it).

The following sections categorize deep KT models according to their fundamental mechanism into sequence modeling, memory augmentation, attention mechanism, graph convolutions, and multi-feature input. Nevertheless, some models fall into multiple categories, trying to combine the advantages of multiple approaches.

2.2.1 Sequential KT models

The usual input of KT models is a sequence of interactions as the student answers a question, gets feedback on whether her answer is correct or not, and moves to the next question. The KT model predicts the probability of a correct answer to the next question which makes KT a sequence prediction problem. Recurrent neural networks (RNNs) are designed to handle sequential data by retaining context (hidden state) across inputs. We can interpret this hidden state as a student's evolving knowledge state and apply RNNs to KT. The downside of this method is the low explainability of the hidden knowledge state because all information is mixed in one vector making it impossible to distinguish knowledge states of individual KCs, or model the strength of the relationships.

Deep Knowledge Tracing model (DKT)

The pioneering model DKT [13] experimented with vanilla RNN and Long Short-Term Memory network (LSTM). In each time step t the recurrent unit of RNN combines the one-hot encoded interaction consisting of KC and its answer correctness $x_t = \{c_t, a_t\}$ and the hidden state from the previous step h_{t-1} to get the new hidden state h_t. The prediction vector y_t is calculated from the hidden state h_t using Multi-layer Perceptron (MLP) with sigmoid activation and represents probabilities for each KC being answered correctly in the next time step $t+1$. LSTM is a more complex form of RNN as it maintains an additional long-term context vector, and thus can work with longer sequences.

DKT achieved significantly better performance in the form of higher accuracy and AUC than the best model at that time, the Bayesian KT model, on three datasets. However, the model cannot handle multi-skill questions (the question index is replaced by the index of the KC required by the question) or include any additional information about the input sequence. Regarding explainability and outputs, for each student, we can visualize evolving probabilities of a correct answer for all KCs, as shown in figure 2.5 taken from the original article. The evolving hidden states can be visualized similarly, not to mention the possibility for LSTM to separately track students' progress in long-term and short-term knowledge bases.

Furthermore, the authors proposed a method how to use DKT to discover latent relationships between KCs and form a directed dependency graph revealing clusters representing higher-level knowledge concepts. It is based on using DKT with RNN core to predict a correctness probability of KC c_i, after observing an interaction with KC c_j. The normalized probability is assigned to the edge between c_i and c_j in the knowledge graph.

There have been several follow-up papers improving the DKT model. DKT+ [14] experimentally shows that DKT fails to reconstruct the input information - sometimes it decreases the KC's probability despite a correct answer regarding the same KC and vice versa. Additionally, they report that the probability prediction for individual skills isn't always consistent and is prone to sudden changes. To remedy these problems, the researchers proposed regularization terms for the loss function: reconstruction error and waviness measures. The first adds a reconstruction error - the difference between the current answer a_t and prediction y_t to the loss function. This should motivate the model to predict a lower probability when the interaction is incorrect. The second regularization aims at smoothing the transitions of the predicted probabilities as it penalizes the difference between y_t and y_{t+1}. DKT+ exhibited more consistent behavior while keeping the performance metrics comparable with DKT.

Another variation of DKT, DKT-DSC, was used in [15], where the authors added student clustering. Before the start of training, students are assigned full learning ability vectors. The student's vector consists of *correct attempts ratio - incorrect attempts ratio* for each skill (including all their interactions). From these vectors, cluster centroids are computed using K-means and they remain fixed from that moment on. At each time interval (consisting of a given number of time steps), the model first assigns students to the learning clusters according to their evolving ability vector (with interactions up to the given time step). Then the training starts - input to the DKT model is extended for the one-hot encoded cluster index the student belongs to at the given time step. That means that DKT-DSC receives additional information about student's long-term performance. Additionally, which cluster the student belongs to can also be valuable information for the teacher.

Sequence KT Summary

In conclusion, recurrent neural networks stood at the beginning of the deep KT. The DKT model [13] showed significantly better prediction accuracy and was further improved in the follow-up papers (DKT+ [14], DKT-DSC [15]). These models offer the visualization of the evolution of predicted probabilities for all questions and the evolution of RNN hidden states, both representing the overall knowledge state of the student. They were studied only in scenarios with items being the KCs (meaning with a low number of items).

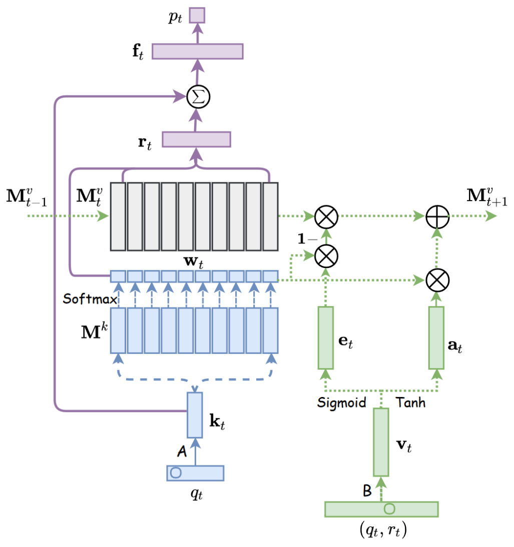


Figure 2.6 The architecture of the Dynamic Key-Value Memory Network from [16]

2.2.2 Memory-augmented KT models

Memory-augmented neural networks for knowledge tracing can more explicitly represent the knowledge state of a student during learning. Moreover, they can model the student's long-term memory for longer sequences. Individual key-value pairs nicely correspond to knowledge concepts and the evolving knowledge states about them.

Dynamic Key-Value Memory Network (DKVMN)

The best-known model in this category is Dynamic Key-Value Memory Network (DKVMN) [16]. It extends memory-augmented NN to contain two memory matrices - static key-memory matrix that contains embeddings of latent KCs, and dynamic value-memory matrix that tracks the evolving knowledge state about the latent KCs of the student. The architecture (visualized in figure 2.6) is the following: during the read process, embedding of the question q_t is compared with the key matrix to get correlation weights - how much is the question related to each of the latent KCs. The weights are used to read the value matrix to get a weighted sum of the knowledge states r_t. The acquired read content is combined with the question embedding to get the final prediction. The next step is the writing process, during which the value-memory matrix is modified according to the question and the observed interaction and the prediction continues with the next question.

The authors showed that DKVMN is able to guess the most relevant KC for most questions. Moreover, even when the number of latent KCs is set to be higher than the ground truth, the model was shown to form the correct number of clusters.

Continuous Personalized Knowledge Tracing model (CPKT)

The work [17] introduces a Continuous Personalized Knowledge Tracing (CPKT) model that aims to further personalize DKVMN. It enhances the architecture with the user embedding vector that is mixed into correlation weight calculation and updates vector computations during the writing process. The goal is to account for each student's differing ability to learn (personalized memorization and forgetting behavior). The authors also proposed an online learning and prediction paradigm based on a stochastic shared embedding that swaps similar item embeddings during training. CPKT builds and updates a transition matrix that captures the probability of a student solving an exercise i after exercise j and samples what embeddings should be swapped from it.

Memory-augmented Attentive Networks (MAN)

Recently, the Memory-augmented Attentive Networks (**MAN**) model was proposed in [18]. It combines the attention mechanism for recently acquired knowledge and memory-augmented NN for long-term knowledge. They are combined with another attention mechanism that weights the trade-off by measuring the correlation between the current exercise and the contextual exercises. The authors described a skill-switching phenomenon, which is a characteristic of a student sequence during which the student returns to the KCs they already practiced in the past after a few interactions with other KCs. MAN's was shown not to degrade with repeated skill switching.

Memory-augmented KT Summary

In summary, DKVMN [16] brought a new mechanism into the field of KT that allowed separate modeling of latent KCs. These latent KCs can be compared with true KCs and through them get better reports on the knowledge state of the true KCs. Usually, we can only get a probability prediction for one question that is used as a query in the dynamic key-value memory. The memory KT models were reported to better handle long student sequences than the sequential models.

2.2.3 Attention KT models

The attention mechanism in KT is used to model the relative importance of past students' interaction with the current prediction. Also, attention can be used for visualization of these relationships, leading to better explainability.

Self-Attentive model for Knowledge Tracing (SAKT)

One of the pioneering works is the Self-Attentive model for Knowledge Tracing (SAKT) [19]. The authors discovered that it is beneficial to give a different amount of consideration to each of the past interactions when predicting the outcome of the next question. The idea is, that each KC is strongly related to only a few other KCs. Therefore we should base our prediction on past interactions involving them. The model starts by embedding the interaction x_t and the next question q_{t+1}. The interaction embedding is added to the position embedding and transformed with the respective matrices into key and value vectors. The question embedding is translated into a query vector. The query and key are merged into attention weights for scaling the values. The weighted sum of values is fed into a NN to get the final prediction.

The authors used attention weights from SAKT to visualize the relative importance of exercises. They summed up and normalized attention weights from all sequences from the synthetic dataset (with known ground truth relationships) and clustered the exercises according to their relative attention. SAKT was able to achieve the perfect clustering of the underlying KCs.

Separated Self-Attentive Neural Knowledge Tracing (SAINT+)

The authors of SAINT [20] further leverage the attention mechanism for KT by applying it first separately on exercises and responses and then combining the information from the two with another attention layer (model architecture is depicted in figure 2.7 on the left). In more detail, the exercise embeddings $e_1, \dots e_t$ are encoded with multi-head self-attention and used as keys and values for the final decoder attention. The shifted response embeddings $S, r_1, \dots, r_{t-1}$ are

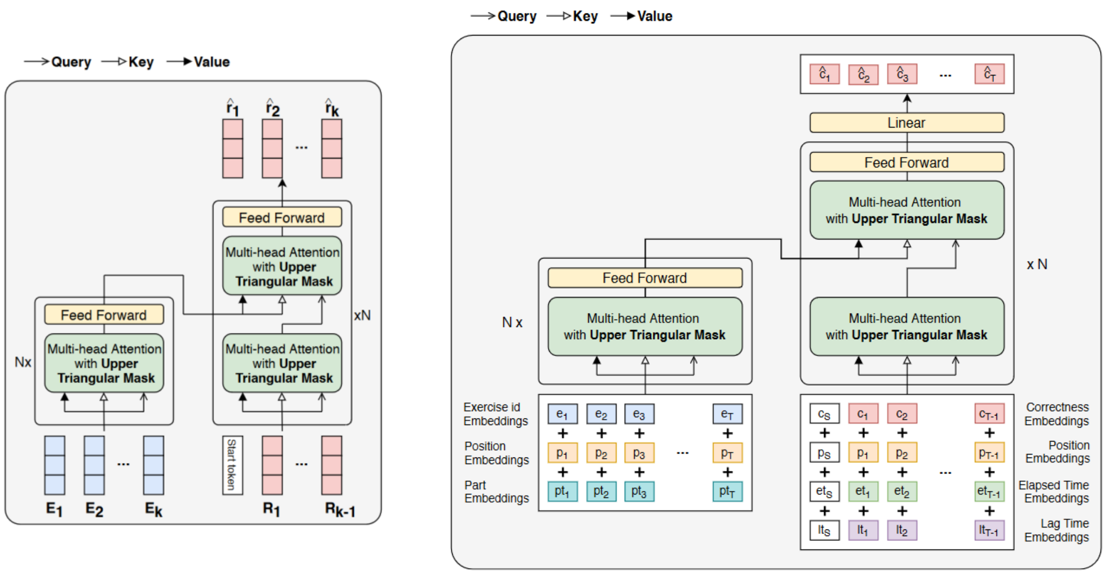


Figure 2.7 The architecture of SAINT [20] (left) and SAINT+ [21] (right)

encoded with another multi-head self-attention and used as queries in the final decoder attention. The final vector for prediction is read by the decoder and combines the information from past interactions and the current exercise.

In contrast with previous works on attention, the attention-encoded correctness of the answers up to the previous step are used as a query, and the attention-encoded questions up to the current step are used as a query. This choice is supported by experimentation with the attention architecture.

To further improve the results, SAINT+ [21] incorporates two temporal features into the response embedding - time the student took to answer the question (elapsed time) and time from the last exercise (lag time). It is done by adding the embeddings of correctness, elapsed time, and lag time together. SAINT+ achieved higher prediction accuracy and area under the ROC curve (AUC) than the compared models that didn't use any additional information apart from the interactions.

Context-Aware Attentive Knowledge Tracing (AKT)

With the goal of improving the interpretability of the deep KT, the authors of [22] proposed a modification of the attention mechanism to include the forgetting behavior of the student. A multiplicative exponential decay is added to the attention mechanism that lowers the attention score of interactions further in the past.

AKT is composed of two self-attention encoders to obtain context-aware representations - one for questions and one for interactions. Each encoder takes the embeddings of the past questions (interactions) and combines them with the previous questions (interactions) with self-attention. The resulting representations are run through a knowledge retriever with monotonic attention that combines the questions and interactions and applies attention weights with the time decay to get a hidden state for the current time step. The prediction is done by passing the hidden state through the feed-forward network.

Another contribution of the paper is Rash model-based embeddings. The authors proposed to embed a question and its knowledge concept together as

$$x_t = c_{c_t} + \mu_{q_t} \cdot d_{c_t}$$

where c_{c_t} is the vector embedding of the KC c_t required by the question q_t, d_{c_t} is a vector that summarizes the all questions that require the KC c_t, and μ_{q_t} is a scalar difficulty of the question q_t. The resulting embedding enables the model to overcome the sparse data problem when using questions as items (and not the required KCs).

Bayesian KT with Attention

The attention mechanism can also enhance the traditional Bayesian KT model as shown in [23], where the authors used the attention mechanism on the learner's attributes (like age, gender, teacher) and her interactions with exercises to improve the prediction of the mastery level. The changes in attention during the learning process suggested that student's attributes are more important for prediction in the beginning of the learning process, whereas the activity on exercises is used for prediction later in the process. Consequently, the authors suggested that using student features could be one of the solutions for the cold-start problem in KT.

Programming Knowledge Tracing

The authors of [24] focus on programming knowledge tracing. KT on programming exercises is especially difficult because it involves multiple skills per exercise and poses a challenge on how to recognize whether the code contains evidence of mastering a certain skill. The model first acquires code-skill interaction embedding with attention. In more detail, the KCs are used as queries, and latent patterns of code are represented as keys and values. To get the embedding, the model averages attention-weighted values of KCs required by the exercise. The core of the model is the LSTM, which uses concatenated multi-hot encoding of skills required by the exercise, one-hot encoded correctness of the answer and the previously described code-skill interaction embedding as inputs. The model outputs the probability of answering the next exercise correctly.

Attention-based KT Summary

The attention mechanism was originally brought into KT to better model the relationships between items and proved to be very effective. Also, the transformer architecture allows better parallelization (as opposed to recurrent NN) which leads to shorter training times.

2.2.4 Graph-based KT models

There is a lot of useful structured information in KT. Namely, what KCs are examined by the question, whether there is a dependency between KCs, or if the learning materials form a hierarchy. This information is often best described by a graph which might be processed by graph neural networks (GNNs). Additionally, graphs can (usually) be visualized conveniently and used for understandable reporting for students, teachers and parents.

Graph-based Knowledge Tracing model (GKT)

The first graph-based KT model was GKT [25]. It reformulated KT as a time series node-level classification task in the GNN. The knowledge graph consists of KCs modeled as nodes (more precisely, node value is the hidden knowledge state about the KC) and the dependencies between them as directed edges. The architecture overview is shown in figure 2.8. The model's input is the interaction at time step t, and it outputs probabilities of answering correctly each KC in the next time step $t + 1$.

The first step is the aggregation, where we concatenate the hidden knowledge states of the required KC and its neighbors with the interaction embedding for the required KC and KC embedding for the neighbors. Then, we update the knowledge states of used KCs by message-passing the aggregated representations of each KC used in the graph. The new hidden states are

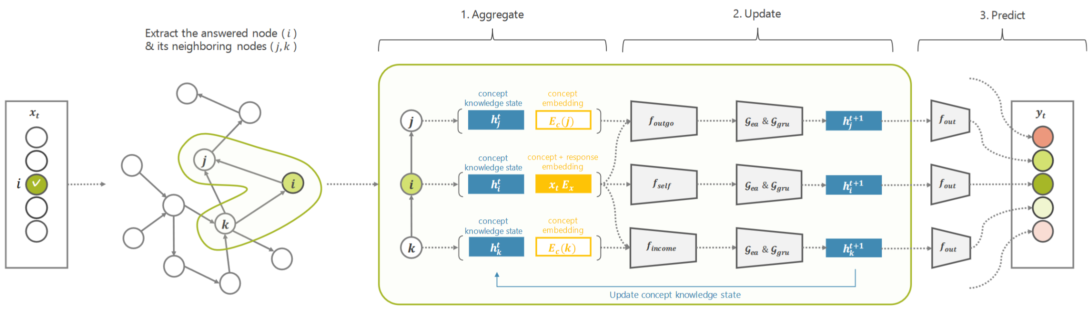


Figure 2.8 Architecture of GKT model from [25]

computed by running the message through the erase-add gate unit, followed by the gated recurrent unit (GRU). Lastly, the probabilities are predicted for all KCs from the hidden knowledge states (that were updated for the required KC and its neighbors, but remained the same for the rest of the graph).

GKT can leverage the additional information included in the knowledge graph. However, the graph isn't always provided. Therefore the authors proposed several methods for obtaining it. Either from a statistics-based approach in the form of a transition graph that weights the edges according to the number of times certain KCs were examined after one another, or from a learning-based approach during which the graph structure is optimized along with the other parameters during learning - in the form of multi-head attention, or variational autoencoder.

Structure-based Knowledge Tracing model (SKT)

The Structure-based Knowledge Tracing (SKT) model [26] explicitly employs two types of edges between KCs - bidirectional concept similarity and unidirectional concept prerequisite. They propose two learning propagation techniques to accommodate the two types of edges - partial propagation for directed edges and synchronization propagation for undirected edges. This extension of GKT causes the KCs to influence each other more intuitively.

Graph-based Interaction model (GIKT)

A different graph is utilized by GIKT from [27]. The model distinguishes between questions and KCs (and allows multiple KCs per question) - it models them with a bipartite graph, where edges connect question-nodes with KC-nodes that are required by the question. Graph Convolutional Network (GCN) is used to extract embeddings of skills and questions.

GIKT first encodes the interaction sequence up to time step $t-1$ by combining question embedding with answer embedding (another novelty of this paper) into exercise representation, passing it through LSTM to get the hidden state. To predict the answer to question q_t, GIKT combines q_t embedding, embeddings of the KCs required by the question, hidden state from LSTM, and representations of relevant historical exercises. The interactions between the mentioned elements are weighted with attention to get question q_t probability prediction. The historical exercises are either chosen as the exercises sharing the same KCs with the current question, or as top-k most similar exercises weighted by an attention function (like cosine similarity).

Graph-based Dynamic Interactive KT model (DGKT)

A recent model DGKT [28] combines all kinds of embeddings together - both static and dynamic embeddings for each student, question and KC. Especially notable are the dynamic embeddings of questions and concepts. The authors explain them as varying difficulty after a possible poor

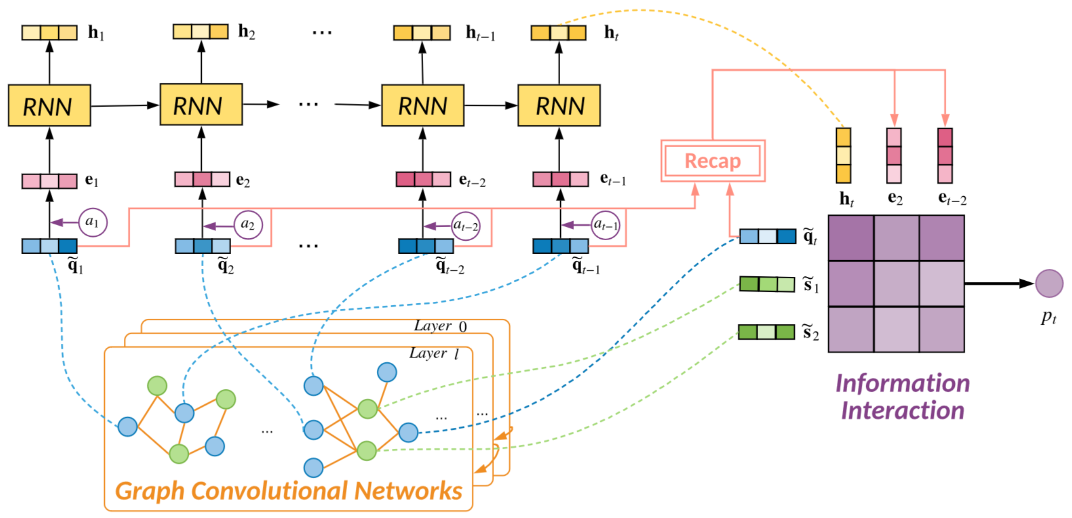


Figure 2.9 Prediction process of GIKT from [27]

explanation from a teacher (students can be confused for a while before grasping the concepts and difficulty returns to a static average). The model works with a heterogeneous graph that contains questions and concepts as nodes and two types of edges - question-concept relationship (KC is required by the question) and concept-concept (a concept is a prerequisite of another concept). The authors propose a pre-training of static embeddings of the questions and concepts with self-supervised graph learning to extract the structural information from the graph. The last highlighted contribution of the paper is influence propagation. It comes from educational theories stating that when a student learns a concept, the knowledge about related concepts also increases.

Concept Map Driven Knowledge Tracing model (CMKT)

The model CMKT from [29] also utilizes a knowledge graph (concept map) that is a directed acyclic graph (DAG) with KCs as nodes and prerequisite relationships between KCs as edges. The topology of the knowledge graph is first encoded using the DAG-GRU network - each KC is one-hot encoded and passed through a GRU cell along with hidden states extracted from the prerequisite KCs' GRU passes. Because the knowledge graph is directed and acyclic, we can construct a network of GRU cells with the same paths of hidden state as the knowledge graph. When there are multiple prerequisites, all hidden states are aggregated together with max pooling. We get an embedding for each KC encoding its position in the graph.

The input of the CMKT model is a concatenated one-hot encoding of a question and aggregated embedding of all KCs required by the question (combined with max pooling). The input sequence is passed through GRU cells and the final prediction is read from the hidden states by a fully connected layer with sigmoid activation. Finally, CMKT utilizes the knowledge graph as a part of its objective function - pairwise relations are extracted from the map and transformed into mathematical constraints for the predicted probabilities.

Inspired by the mastery learning theory, the authors define two soft constraints. First, if the student mastered a certain KC, they had already mastered its prerequisites - so the probability of answering the prerequisite KC correctly should be larger than the probability of the main KC. Similarly, the second constraint assumes that if the student has not mastered a prerequisite KC, they could not have mastered the main KC either - in other words, when the probability of answering the prerequisite KC correctly is low, the probability of answering the dependent KC is even lower. These constraints are included in the objective function where they penalize their violation.

Graph-based KT Summary

To summarize, utilizing the graph structure of the KCs proved to improve the performance of models on the KT task. The core of the knowledge graph constists of knowledge states of KCs as nodes and prerequisite dependencies between them as edges. However, a lot of authors experimented with adding new node types (for example GIKT [27] and DGKT [28] for questions) and edge types (like undirected similarity edges from SKT [26], or question-KC relationships from GIKT [27] and DGKT[28]). The recurrent neural networks remain the heart of the graph-based KT methods having the question or KC embedding of the knowledge structure as input. The hidden knowledge state in the nodes is either static and queried with graph convolutions (GIKT [27], DGKT [28]), or dynamic and updated through message passing and read in each time step (GKT [25], DGKT [28]).

2.2.5 Multi-feature KT

The standard knowledge tracing task is to predict the probability of the student answering the next question correctly only from previous interactions. However, multiple research works showed that including other learning features, such as question text, answering time or student's learning ability, can significantly boost performance.

Multiple Feature Attention Enhanced DKT (MFA-DKT)

The model MFA-DKT from the paper [30] takes full advantage of features available in the Assistments 2009 dataset [31]. It takes 24 features for each time step (including response time, number of hints, or type of first action) and embeds them into denser feature space with Principal Component Analysis (PCA). The embedding is passed through LSTM to get hidden states for each time step up to the current time step t. The hidden states are weighted with attention and combined with feature embedding for the next time step $t+1$ to get the final prediction.

Knowledge Tracing based on multi-feature fusion (KTMFF)

Model KTMFF from [32] adds question text, knowledge concept difficulty, student's ability, and duration time to DKVMN [16] model. Whereas DKVMN compares only the embedding of the question number with the latent knowledge concepts, KTMFF combines the question text feature (extracted by a transformer), question difficulty (calculated as a balanced (weighted?) question's correct answer rate), and the aforementioned question number. The features are put together via a multi-head self-attention mechanism creating a question semantic representation vector.

Another improvement is including a student's ability into the knowledge state vector that was read from the dynamic key-value memory. The ability vector is calculated from the weighted rates of the difficulties of the answered questions. Therefore, the final prediction is made from a vector that combines the read knowledge state, question semantic representation vector and student's ability vector. The last modification is considering the time used to answer the question. The duration time feature is added in the writing process when calculating the new knowledge states of the latent KCs.

Multiple Learning Features-enhanced KT (MLFKT)

The paper [33] took a systematic approach towards multi-feature KT. The authors first categorized features into three types: learner features F_L (like grade, or ability), resource features F_E

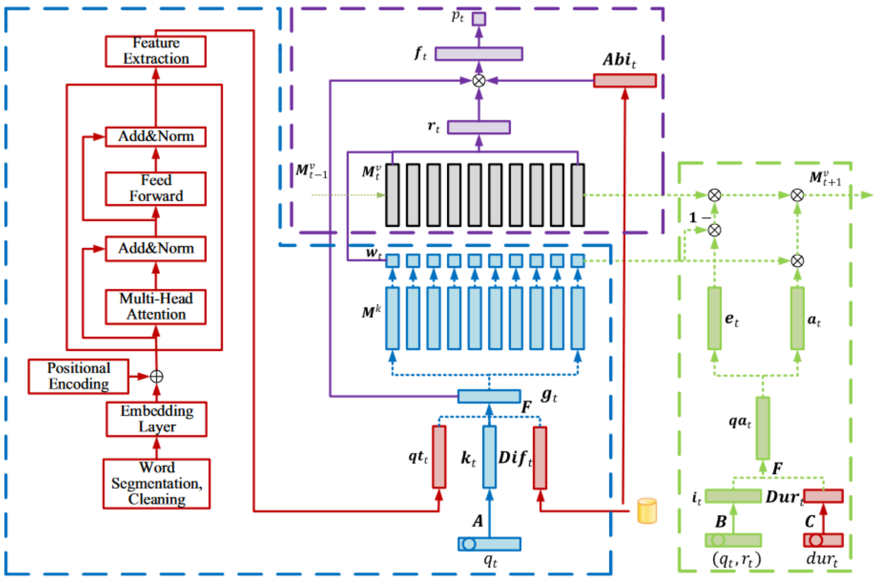


Figure 2.10 The architecture of KTMFF model from [32]. Red parts are additions to the original DKVMN model from [16].

(knowledge points gained by solving the exercise, question difficulty), and response features F_R (correctness, hints used, response time, number of attempts). Then they defined learner-resource response (LRR) as a triplet $\{f^l, f^e, f^r\}$ which can acquire any value from a Cartesian product $F_L \times F_E \times F_R = \{(f^l, f^e, f^r) \mid (f^l \in F_L) \wedge (f^e \in F_E) \wedge (f^r \in F_R)\}$. The authors hand-pick LRR channels that will be used as inputs according to Item Response Theory and Psychometric Theory.

The proposed model MLFKT first combines the three features from each LRR channel into one cross-feature. The cross-features are one-hot encoded and passed through a stacked auto-encoder. The representation of LRR channels are assigned weights using the attention mechanism and passed into bidirectional LSTM. The Bi-LSTM is supposed to track the long-term memory of the student and account for her forgetting behavior.

Multi-feature KT Summary

To summarize, multi-feature KT aims at improving the performance and explainability of KT models by incorporating more diverse information into the modeling process than the simple sequence of interactions. The extra features are either merged together with a dimensionality reduction method (such as PCA in MFA-DKT [30], or attention in MLFKT [33]), or integrated into the model mechanics (for example in KTMFF [32], question text and difficulty influence the reading, student's ability influences the final prediction from the read value and the duration of the answer is considered during the write process). Due to the extreme variability of the features, preprocessing is usually needed. One-hot encoding is popular, especially with end-to-end learned embedding or autoencoder embedding. Furthermore, visualization of feature embeddings can provide us with meaningful insights into the learning process.

Chapter 3

Knowledge Tracing Datasets

Knowledge tracing predicts student's future study performance based on past interactions with the education platform. The fundamental units of information in the KT datasets are these interactions (action logs) - for example, submitting an answer, selecting the next learning material, or displaying a question. The key component is an answer to a question with the evaluation of whether the answer was correct. However, datasets usually contain more types of interactions or additional action log metadata (interaction features).

The most important characteristic of the KT datasets is the number of interactions per item. It determines the amount of information the model gets about the item. Moreover, for most KT models, it is the only information about the individual items - in what context the item appears, how many students answered it correctly with what background, and whether it is similar to another item.

The table 3.1 states the most important metrics of the most commonly used KT datasets. The datasets Assistments 2009, EdNet KT1 and Marast were selected for the usage in experimental part of this thesis and therefore downloaded and studied directly. The metrics of the rest of the datasets were taken from other papers ([34]) and public descriptions in the datasets' source articles.

Dataset	Subject	Year Gathered	#Interactions	#Questions	#KCs	#Students
ASSISTments2009	Math	2009-2010	346 860	26 688	149	4 217
ASSISTments2012	Math	2012-2013	6 123 270	179 999	265	46 674
ASSISTments2015	Math	2015	708 631	100	x	19 917
ASSISTment Challenge	Math	2004-2006	942 816	3 162	102	1 709
EdNet KT1	English	2017-2019	95 293 926	13 169	188	784 309
EdNet KT2	English	2018-2019	56 360 602	13 169	188	297 444
EdNet KT3	English	2018-2019	89 270 654	13 169	293	297 915
EdNet KT4	English	2018-2019	131 441 538	13 169	293	297 915
EdNet Kaggle	English	2017-2019	101 230 332	13 782	1 519	393 656
STATICS2011	Engineering Statics	2011	361 092	1 224	85	335
Junyi Academy	Math	2012-2015	25 925 992	722	41	247 606
Junyi Kaggle	Math	2018-2019	16 217 311	1 330	10/42/171	72 630
Algebra 2005	Math	2005-2006	813 661	1 084	112	575
Algebra 2006	Math	2006-2007	2 289 726	90 831	523	1 840
Bridge to Algebra	Math	2006-2007	3 686 871	19 258	493	1 146
Marast	Math	2016-2024	240 656	2 382	12 263	56

Figure 3.1 Table of knowledge tracing datasets used in research with their main metrics

ASSISTments2009	https://sites.google.com/site/assistmentsdata/home/2009-2010-assistment-data?authuser=0
ASSISTments2012	https://sites.google.com/site/assistmentsdata/datasets/2012-13-school-data-with-affect
ASSISTments2015	https://sites.google.com/site/assistmentsdata/datasets/2015-assistments-skill-builder-data?authuser=0
ASSISTment Challenge	https://sites.google.com/view/assistmentsdatamining
EdNet	https://github.com/riiid/ednet
EdNet Kaggle	https://www.kaggle.com/competitions/riiid-test-answer-prediction/overview
STATICS2011	https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=507
Junyi Academy	https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=1198
Junyi Academy Kaggle	https://www.kaggle.com/datasets/junyiacademy/learning-activity-public-dataset-by-junyi-academy
KDD Cup (Algebra datasets)	https://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp
Marast (system website)	https://marast.fit.cvut.cz/

3.1 ASSISTments

ASSISTment is an online math tutoring platform that aims to connect teachers and students and provide them with personalized and responsive educational tool. The teachers prepare sets of exercises and students get immediate feedback while completing them. There are multiple versions of the dataset, collected at different times with different goals. They vary in available features as well as the granularity of the activity logs. It is important to note that students can opt to ask for a hint during question answering which is reported as a wrong answer. Consequently, even though the student is able to answer the question correctly using hints, it results in an incorrect answer log.

3.1.1 ASSISTments 2009

The first, and probably the most widely used, version is the ASSISTments 2009 dataset [31] (used by for example DKT [13], or GKT [25]). It was gathered during the school year 2009-2010.

It is divided into two parts: skill builder data and non skill builder data. The skill builder (or mastery learning) is a mode of the ASSISTments platform in which the student is asked to master a certain skill. The student is considered to master a KC when they correctly answers three questions regarding that KC in a row. The assignment is then considered complete and the student continues to another problem set.

ASSISTments questions can require multiple KCs - in the original dataset this was recorded with duplicated logs for one interaction. Therefore, researchers prepared a collapsed version of the skill builder dataset which is recommended for the KT research. The questions requiring multiple skills are assigned a collapsed skill ID consisting of original skill IDs connected with an underscore. Usually, these combinations are categorized as an additional skill. From now on, I will consider only the recommended skill builder, corrected, collapsed version of the ASSISTments 2009 dataset.

The core features of this dataset are *user_id* as student identifier, *problem_id* as question identifier, *skill_id* as the identifier of the KCs, and *correct* as the binary evaluation of the answer. As mentioned earlier, the interaction is evaluated as correct only if the student answers the question correctly and on the first try, without using any hints. Additionally, the order of interactions is determined by *order_id*. The only textual feature is the *skill_name* which is a short description of the KC - for example: ”Complementary and Supplementary Angles”, ”Least Common Multiple”, ”Write Linear Equation from Graph”, or ”Histogram as Table or Graph”.

There are 149 unique KCs also counting the combinations - because this version of the dataset has collapsed skills. The taking KC combinations as a new KC is a common procedure in knowledge tracing, because most of the KT models cannot handle multiple KCs per question. In the case of the ASSISTments 2009 dataset, the real number of KCs is 207.

Other features that might help with the prediction are *ms_first_response* (longer response times could suggest student's uncertainty, and too quick answers might be guesses), *first_action* (whether it was an attempt or ask for a hint), and *hint_count* (the number of hint the student got for the current problem). Regarding the student's background, we get only *teacher_id*, *school_id*, and *student_class_id*.

One interesting characteristic that I will later use for baseline implementation is that students tend to have more correct answers in a row – specifically, 72% correct answers are followed by another correct answer. For completeness sake, the chance of getting a correct answer after an incorrect one is 50%, so it doesn't give us any additional information.

The target column *correct* is slightly imbalanced at 65% of the correct answers. ASSISTments 2009 contains circa 350 000 interactions, 27 000 questions, and 150 KCs which gives questions and KCs significantly different numbers of interactions per item. The average number of interactions per KC is 1900 with 75% of KCs having more than 234 interactions. In contrast, questions have on average 16 interactions, with 75% of the questions having under 16 interactions.

Preprocessing

As one of the most widely used datasets, I have selected ASSISTments 2009 for the experimental phase for KT model testing. I downloaded the official collapsed skill builder dataset from the official website[1]. The dataset comes in csv format. To prepare it for use, I have first removed the logs without *skill_id* or target value *correct*. Then, students with less than 5 interactions has been filtered out. I have enumerated the *skill_id* and *problem_id* so they form a consecutive sequence from 0 to the number of unique skills - 1 and number of unique problems - 1 for the later part (while maintaining the order of items). Next, I have created the interaction encoding used in most KT models:

```
encoding = item_id + correct * num_items
```

where *correct* is the binary indicator of the correctness of the answer. I have computed the encoding for both questions (problems) and KCs (skills). Finally, the interactions have been sorted according to *order_id* and grouped according to the *user_id* into student sequences for use. After this procedure, 281 890 interactions of 3 644 students remained in the dataset. The testing part of the dataset (consisting of 20% of the sequences) was separated at this point.

3.1.2 Other ASSISTments datasets

All datasets provided by the ASSISTments platform are conveniently listed at e-trials website[2]. I will mention only the most commonly used ones.

The 2012 ASSISTments dataset focused on affect prediction and was introduced in [35]. It is the largest ASSISTments dataset. However, most questions don't have any KCs assigned to them. The dataset consists of the same feature columns as ASSISTments 2009 with additional affect columns - average confidence of the student being frustrated, confused, concentrating, and bored. The emotions were predicted by a detector from [36] and later used as a target in follow-up works (namely [35]).

Another version is the ASSISTments2015 dataset. Nevertheless, it contains only four attributes: sequence ID (question set ID), user ID, correctness of the answer, and log ID (determining the order of the interactions). It only contains responses on 100 problem sets with the highest number of student responses.

The ASSISTments Longitudinal Data Mining Competition 2017 contains the most descriptive information about interactions - including several affect metrics, duration of actions and pre-computed learning performance metrics (like Bayesian KT estimate, average correctness or number of hints used so far). The purpose of the competition was to follow students from middle school, through college and into their careers.

3.2 EdNet

EdNet [37] is an enormous hierarchical dataset collected from an online tutoring platform Santa[3] aiming to prepare students for the Test of English for International Communication (TOEIC) exam. It offers four data levels, each containing different types of student activities that bring a more detailed view at the learning process with a more granular action log. Questions in EdNet are structured into bundles - a set of questions sharing the same learning material (like reading passages, pictures, or listening recordings).

KT1 captures the basic question-answering including question ID, bundle ID, correctness of an answer, and elapsed time. KT2 adds more information about students interacting with the

[1]link to google disk with the data:`https://drive.google.com/file/d/1NNXHFRxcArrU0ZJSb9BIL56vmUt5FhlE/view`

[2]https://www.etrialstestbed.org/resources/featured-studies/dataset-papers

[3]https://www.aitutorsanta.com/

questions (for example, alternating between answers). A column *action_type* is added with three types of actions: enter (when the student is first shown the bundle), respond (when the student inputs an answer), and submit (when the student submits her final answers to the bundle). Also, a source of the question is recorded - whether the question was recommended based on a sprint mode (student chose a topic she wants to practice), daily recommendation, adaptive offer (the number of incorrect answers regarding a certain topic exceeded a threshold and the student is given an exercise practicing that topic), or it is a selection from the Santa AI tutoring system. KT3 extends the learning activities with reading the expert's commentary and watching video lectures. This provides information about other learning opportunities the student has. The last addition is the KT4 dataset which includes interaction with video and audio recordings (pausing and playing video/audio) and subscription information (pay, refund, enroll coupon).

For experimental phase, I have used a version of the EdNet dataset from Kaggle[4]. It offers the data in a more suitable format - in single csv file, as opposed to a csv per student from the original repository.

Similarly to the ASSISTments dataset, EdNet's questions has tags assigned to them (5 787 unique tags) - even though no text description for them is available, they are closest to the KCs and I have used them as such. As each question can require multiple KCs, the combination is (in a similar way as described earlier) encoded as a new KC resulting in 1 519 KCs. The questions have 3.8 KCs assigned on average.

The Kaggle version contains two types of interactions - questions and lecture watching differentiated by *content_type_id*. However, only 2% of the logs are reports of lectures (which still totals 1 959 032 interactions). Another potentially useful feature is *prior_question_elapsed_time* - the average time in milliseconds the student took to answer each question in the previous bundle (a set of questions sharing the same material). Additionally, EdNet offers *part* feature describing the part of the TOEIC test that the question exercises (each focused on a different language skill: listening with question-response, conversations, or talks, or reading with incomplete sentences, or text completion). EdNet does not contain any textual features.

The target distribution is similar to the ASSISTments with 66% of the correct answers (to question logs). Also, the chance that a correct answer will follow a correct one is 69%, but in contrast with ASSISTments, there are more correct answers following an incorrect one (not aligning with the baseline built on predicting the last answer). Because EdNet is so large, it doesn't suffer from the lack of interactions per item. In the part I have randomly selected for implementation, 75% of questions have more than 2 002 interactions with a mean of 4 987 per question.

Preprocessing

The data from Kaggle are split into 5 csv files. I have used *train.csv* to obtain the interaction logs and *questions.csv* to get the question tags. First, I have dropped lecture rows (those with *content_id != 0*) leaving only interactions with questions. I have merged the interactions with questions to connect question ID with the KC (tags). Next, I have enumerated the question ID and KC ID and created interaction encoding (the same as in the ASSISTments 2009 dataset). I have sorted the interactions according to the timestamp. Lastly, I have grouped the interactions according to the student and filtered out students with less than 5 interactions. The dataset has proved too large to be handled with standard computational resources, so I have selected a random partition of 30% of the students (aprox. 118 000 sequences from the set of 390 000 sequences). Finally, I have separated 20% of the selected sequences for testing.

[4]https://www.kaggle.com/competitions/riiid-test-answer-prediction/data

3.3 Marast

Thanks to Ing. Tomáš Kalvoda, Ph.D. from FIT CTU I have been given access to data from the math quizzes from the faculty's system MARAST [5]. The students get quizzes assigned by the teacher through MARAST or they can exercise in the practice mode. All exercises are created by teachers and there are two types of them: multi-choice and open-text.

In the multi-choice exercise, the student is given a question and four statements. The student is supposed to decide which of them are true and which are false. The answer is evaluated as correct only if all four statements are answered correctly. Otherwise, the student has to wait a given time interval before they can continue to the next question. This is to prevent students from guessing repeatedly.

In the case of an open-text question, the student is given a task and has to come up with a solution, which is usually a mathematical expression or a number. All answers are evaluated automatically, therefore solutions must be in a closed form.

This dataset has two major advantages for knowledge tracing as textual features and non-binary correctness evaluation are present. Text features are rare and limited in the KT datasets - to my best knowledge, only Assistments 2009 contains a short description of the Kcs and aside that, I have not found any KT with textual features. Therefore, full text of questions promises new opportunities and allows further experimentation. Also, a short description is provided, both in Czech and English.

The non-binary answer evaluation can be extracted from the multi-choice questions. The dataset contains information regarding which statements the student marked as true and which were correct. From that, I have been able to extract a (0, 0.25, 0.5, 0.75, 1) scale that could give the KT models further information.

For this dataset, I have used topics as knowledge concepts. The data comes from the course ”Elements of Calculus” which include topics like ”Manipulation of algebraic expressions”, ”Bijection, injection, surjection”, ”Tangent line to a graph” or ”Integral criterion”. The concepts have a shallow hierarchy with 8 parental KCs. Students in this course often focus on one part of the subject's curriculum and practice it on multiple exercises. This results in jumping between KCs - a sequence with a few different KCs that are randomly mixed up with approximately the same number of occurrences of each KC.

Preprocessing

The data were originally in JSON files with topics (KCs) and problems (questions) in separate files. I have transformed the interaction logs into a pandas DataFrame. The target *correct* column was set to either a binary indicator for open-text questions or the mean of four binary indicators of the correctness of the statements from the multi-choice question. Next, I loaded the csv with questions while preprocessing the question texts: I have transformed the latex math notation into plain text, removed escaped white spaces, and translated texts into English. I have done the translation because most state-of-the-art language models are for the English language. Finally, I have merged the questions with interaction logs. As a last step, I have grouped the logs according to the *user_id* and separated the test set.

3.4 Specification of Knowledge Tracing Dataset

This section aims to help developers of online educational platforms decide what data to gather, so the dataset is suitable for knowledge tracing with the best possible results.

A key components of each KT dataset are interactions - tuples of student ID, item ID, correctness of the answer, and a timestamp (for determining the order of interactions in time). Item

[5]https://marast.fit.cvut.cz/

is any unit of content that can be evaluated as correct or incorrect - those are most commonly questions, quizzes, or exercises. Most KT models use binary correctness evaluation (correct/incorrect), however a more granular scale could give the model valuable additional information. For example, continuous scale from 0 to 1 representing the percentage of the task done correctly could be used. Alternatively, even using another ML model to evaluate the student's answer and obtain a more descriptive vector from it might be recommended - nowadays, large language models (LLMs) show promise to make this work very well.

Items can be part of a hierarchical structure - knowledge tracing works mostly with questions (exercises) that require an understanding of one or more knowledge concepts to be answered correctly which gives us two levels of detail. Additionally, items usually come with relationships between each other such as similarity of KCs (or questions), or denoting that an item is a prerequisite of another one. The graph structure of items is an area of knowledge tracing research and the section 2.2.4 describes its utilization.

Apart from the previously mentioned key features, time plays an important role in a learning path - time spent solving the exercise can indicate a student's skill confidence, or time passed between study sessions might be leveraged by the models to estimate the forgetting behavior. Also, learning seldom comes only in the form of questions and quizzes, the student normally takes part in classes, goes through reading material, or watches video lectures. For the sake of future KT models, it is beneficial to gather as much information about these study opportunities as possible.

Nowadays, LLMs push the boundaries of language modeling and are used in many real-world tasks. Therefore, textual features are becoming the most important part of the KT datasets. The text of questions and their solutions, student's answers and the learning material could leverage the sequence prediction to rich knowledge state modeling.

Chapter 4
Implemenation

The primary goal of this thesis is to investigate the application of knowledge tracing within a personalized educational application. Therefore several KT models were implemented for experimentation and analysis. During describing the implementation, I focused on the intuition behind the inner workings of each model rather than on the mathematical definitions to investigate the explainability of the model.

All implementations are in Python and in the PyTorch library. Having in mind the future developers of the educational application, I have created a library EduKT with the datasets and models I studied. I either copied and adapted the code from the published repositories of the original authors (to ensure authenticity) or implemented the model as close as possible to the paper. The goal was to analyze and compare the models under the same conditions and in the same environment.

4.1 Datasets

The preprocessing of the datasets was previously outlined in Chapter 3 within their respective sections. Each dataset presents a distinct set of features, introducing unique challenges for the knowledge tracing models. These datasets were integrated into the EduKT library, aiming for a standardized interface across all of them. Leveraging PyTorch, I structured the datasets as classes inheriting from *torch.utils.data.Dataset*, adhering to PyTorch's dataset interface.

Each dataset offers three distinct modes: *questions*, *kc*, and *all*. The *questions* and *kc* modes facilitate the most common inputs and outputs required by the models, comprising plain item sequences with item IDs encoded with their correctness. Although the underlying data and separated test sets remain consistent across all three modes, the dataset simplifies the various access needs. This enables seamless model comparison, allowing direct performance evaluation of models trained on questions, KCs, or other features within a single dataset.

The *all* mode provides access to all available features specific to each dataset, which were selected and prepared during preprocessing. For instance, the Assistments dataset offers supplementary features:

- *all_kc* = in Assistments questions may be associated with multiple KCs, their IDs can be extracted from this feature
- *kc_text* = short description of the KC, if the KC is a combination of multiple original KCs, their text was concatenated with "+" and included in the feature
- *response_time* = indicates the duration taken by the student to respond to the question (in milliseconds)

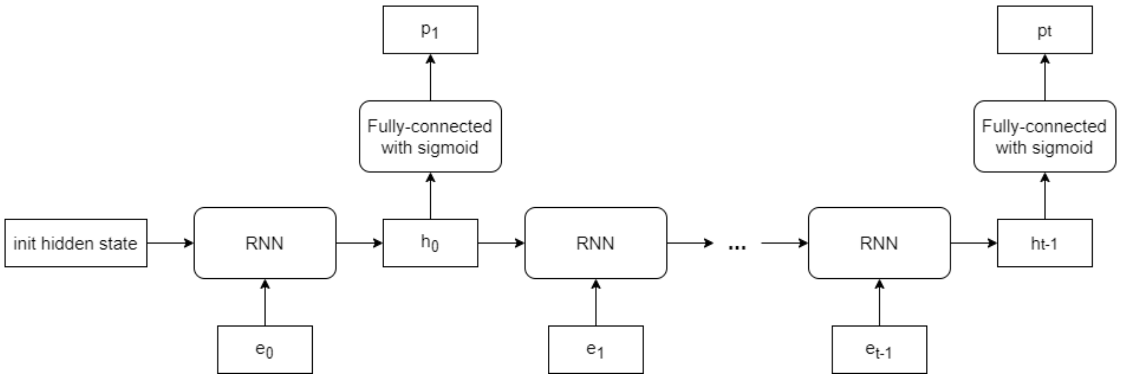


Figure 4.1 Prediction process of DKT-RNN

Conversely, EdNet lacks textual features but provides data on multiple KCs assigned to a single question (*all_kc*) and the time elapsed during a student's response (*response_time*).

The newly introduced Marast dataset encompasses a comprehensive range of textual features, including:

- *answer_text* = text provided by the student as an answer, usually a mathematical expression
- *question_text* = full text of the question translated from Czech to English and with LaTeX expressions textually represented
- *kc_text* = short description of the KC

Moreover, all datasets implement either left-side or right-side padding of the sequences to facilitate seamless batching with PyTorch's *DataLoader*. Additionally, an optional sequence shift functionality is provided. This feature enables the sequences used for models' predictions to be shifted one step back to $0, \ldots, t-1$ (e.g., item encoding or textual question and KC descriptions), while sequences containing answer information and data for evaluating the model's predictions can be shifted one time step ahead to $1, \ldots, t$. This simplifies the prediction code for most models.

4.2 Deep Knowledge Tracing (DKT)

DKT [13] is the most fundamental deep knowledge tracing model, so I selected it as a representative of the sequential KT. The original published code was in Lua, so I implemented both variants from scratch. The core of DKT is a recurrent neural network - either vanilla RNN or LSTM that sequentially takes in the interaction sequence and predicts probabilities of all items in the next time step. To be more precise, the input is a sequence of encoded interactions for time steps $0, \ldots t-1$. The interaction is a number calculated from the item id m and the binary correctness of the answer to that item a with the following formula:

$$e = m + a \cdot |M|$$

with $|M|$ being the total number of items in the dataset and e the encoding number of the interaction. The output is a sequence of vectors containing predicted probabilities for all items for time steps $1, \ldots t$. That means that DKT predicts probabilities of all items in time step t from the interaction sequence up to the time step $t-1$. The whole process is depicted in figure 4.1 where RNN can be either a vanilla RNN cell or an LSTM cell.

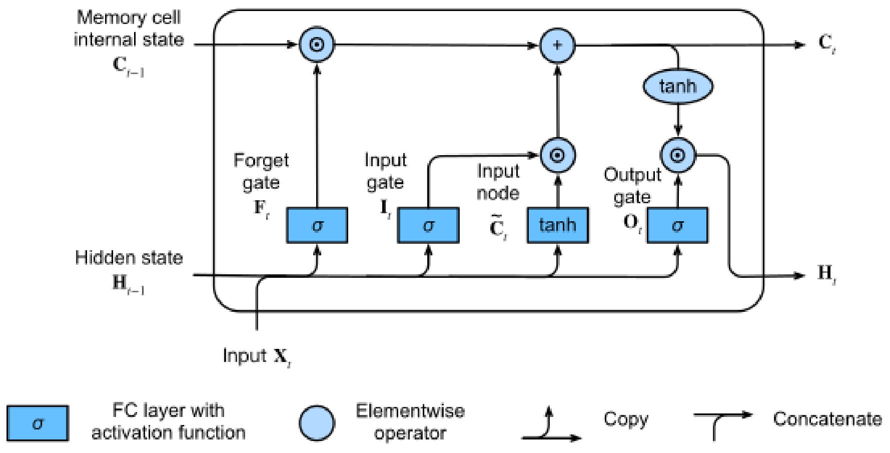


Figure 4.2 Architecture of LSTM cell, taken from [38]

During prediction, all information about the previous interactions is stored in a hidden vector h_t that is updated in each time step. This vector can be interpreted as the student's inner knowledge state and it is evolving as the student learns.

DKT-RNN is a model with vanilla RNN in its core. In each time step, the encoded interaction is embedded into e_t and combined with the previous hidden state h_{t-1} into the new hidden state h_t according to the following formula:

$$h_t = \tanh(e_t W_{eh}^T + b_{eh} + h_{t-1} W_{hh}^T + b_{hh})$$

where W_{eh} and W_{hh} are trainable weights that transform the input vectors e_t and h_{t-1}. Originally in the paper, the interaction was directly passed as a one-hot encoded value to the RNN, but I replaced it with a trainable lookup table embedding because the one-hot encoding of the batched input was causing memory problems for datasets with a large number of items. The multiplication with matrix W_{hh} simulates the evolution of the inner knowledge state in time - the interaction of the vector components (units of knowledge), the forgetting behavior, or the sinking in understanding. The last part of computation are the trainable biases b_{eh} and b_{hh} which are optimized to simulate the long-term effect on the hidden state - some components tend to drop, others tend to rise. All these parts are in the end added together to form the new hidden state h_t.

The final prediction for the next time step $t+1$ is calculated with a fully-connected layer from the hidden state h_t. A prediction for each item is calculated as a weighted sum of all components of h_t (and scaled between 0 and 1 with sigmoid activation).

In the first time step, the model combines the first interaction e_0 with the initial hidden state which is a zero vector. That means that each student starts with the same "knowledge state" and the personalization happens during the sequence processing (according to the performance on the questions).

The slightly more advanced version of DKT uses LSTM for modeling the student's knowledge state. In contrast with DKT-RNN, it maintains two hidden states - short-term and long-term vectors. The information from the long-term hidden state is explicitly erased by a forget gate and added by the product of the input node (what to add) and input gate (how much of that should be added). The new short-term vector is a combination of the long-term vector and the product of the output gate. The short-term vector is then used for prediction. A schema of LSTM is shown in figure 4.2.

The DKT-LSTM has more trainable parameters than DKT-RNN - each gate has its interaction weight matrix (working as an embedding) and previous short-term hidden state weight matrix (totaling in 4 matrices - 4 distinct ways of combining the interaction with the inner knowledge state).

Overview of DKT

Input:	Interaction sequence $e_0, \dots e_{t-1}$
Output:	Probability for each item to be answered correctly in the next time step t
Mechanics:	Recurrent neural network
Knowledge state:	Hidden state of the RNN

The paper [13] experimented on self-prepared simulated data, Khan Academy dataset (not published), and Assistments 2009 [31]. The authors worked only with KCs associated with questions to prevent over-parametrization of DKT (the number of KCs is significantly smaller than the number of questions).

The only model-specific hyper-parameters of DKT are the dimension of the hidden state and the embedding dimension of the item. In theory, a larger hidden vector gives the model more space to store information but can lead to overfitting and requires more data to be trained properly. For visualization purposes, both versions of DKT return a sequence of hidden state vectors.

4.3 Dynamic Key-Value Memory Network (DKVMN)

As a representative of the memory-based KT I selected DKVMN [16]. The implementation was adapted from the publicly available repository [1]. DKVMN is designed around a static key memory coupled with dynamic value memory. Its primary objective is to infer latent KCs and model the student's knowledge state with them. The number of latent KCs is given by the number of memory slots in DKVMN, they are described by their static key vectors and have a dynamic knowledge state in the form of the value vector. An overview of DKVMN is depicted in figure 2.6.

At each time step, DKVMN receives an encoded item number, compares it with the latent KCs in its memory, combines their knowledge states into a knowledge state for the given item, and transforms it into the predicted probability. Then (after the student answered the question and their answer was evaluated), DKVMN gets an encoded interaction (containing the correctness of the answer) and updates the dynamic value memory with the information from it.

To describe the whole process in more detail, the value memory is initialized with random values drawn from a normal distribution. The sequential prediction process starts with embedding the encoded item number m_t into a key vector k_t. Subsequently, correlation weights are calculated for that embedding by evaluating its similarity with each of the static key memory slots $M^k(i)$ (embeddings of the latent KCs):

$$w_t(i) = softmax(k_t^T M^k(i))$$

The weights are used to read the current knowledge state r_t regarding the KC required by the question from the dynamic value-memory matrix M_t^v as the weighted sum of its entries:

$$r_t = \sum_{i=1}^{N} w_t(i) M_t^v(i)$$

The resultant read vector r_t is concatenated with the key vector k_t and propagated through a fully-connected layer to produce a summary vector f_t. Subsequently, f_t is passed through another fully-connected layer with sigmoid activation to yield the predicted probability p_t.

[1]https://github.com/lucky7-code/DKVMN/tree/main

The writing process aims to update the memory with the current interaction. In a real-world implementation, the process would be the following: the student got a question to answer, DKVMN predicted the probability of answering it correctly, the student answered and the answer was evaluated and now it will be used to update the inner knowledge state of DKVMN. The first step of the writing process is embedding the encoded interaction into a knowledge growth vector v_t. Then it is decided, what information to forget and what to add to the memory. We get the erase vector e_t by passing the vector v_t through a linear layer with sigmoid activation and an add vector a_t with a tanh-activated linear layer. Each slot of the value-memory matrix is first multiplied by weighted erase vector $1 - w_t(i)e_t$ and then by weighted add vector $w_t(i)a_t$. After this, DKVMN is prepared for the next time step.

Overview of DKVMN

Input:	Item sequence and interaction sequence
Output:	Probability of answering the given item correctly
Mechanics:	Read and write to the key-value memory
Knowledge state:	Static key-memory matrix with dynamic value-memory matrix

In the paper [16], DKVMN was tested on a synthetic dataset from DKT paper [13], Assistments 2009, Assistments 2015, and Statics 2011. The model outperformed BKT, DKT and MANN. The authors of DKVMN worked only with KCs (not questions) to prevent over-parametrization of the model.

Hyper-parameters of DKVMN are:

- embedding dimension of items which is also the dimension of static key memory vectors
- embedding dimension of interactions which is also the dimension of dynamic value memory vectors
- number of memory slots (number of latent KCs)
- dimension of the summary vector

DKVMN relies on the interpretability of the latent KCs which are represented by the memory slots. In the implementation, both the static memory keys and the corresponding dynamic memory values are returned. Lastly, a list of weights is provided to indicate the extent to which each latent KC contributed to the prediction.

4.4 Self-Attentive Model for Knowledge Tracing (SAKT)

SAKT [19] uses the self-attention mechanism to give a different amount of consideration to each of the past interactions. I wrote the code for this model myself, according to the paper. The model takes a sequence of interactions $e_0, \dots e_{t-1}$ and a shifted sequence of items $m_1, \dots m_t$ and outputs a sequence of predicted probabilities for the given items.

First, item, interaction and position sequences are embedded with trainable lookup tables (separate for each element) into matrices M, X, and P respectively, all with size $|seq| \times d$ ($|seq|$ being the max length of the sequence, d being the embedding dimension). Interaction embedding and positional embedding are summed together into an input matrix $\hat{X}$. Next, we transform the embeddings into query, key, and value matrices. The input matrix is linearly projected into key and value matrices, the item matrix is linearly projected into a query matrix (all weight matrices $W \in R^{d \times d}$):

$$Q = MW^Q, \; K = \hat{X}W^K, \; V = \hat{X}W^V$$

Next comes the most important part of the model - query, key and value sequences are passed through multi-head attention. According to the paper, I use the scaled dot attention:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d}})V$$

Multiple attention heads are implemented with separate query, key and value matrices for every head. Their results are concatenated together and combined with a linear projection ($W^O \in R^{hd \times d}$):

$$Multihead(M, \hat{X}) = Concat(head_1, \dots head_h)W^O$$

The result is the weighted sum of values. To prevent label leaks, a triangular (look-ahead) mask is applied. Additionally, SAKT uses a residual connection to propagate the query (given question) information, so attention output is added to the query and then passed through the normalization layer.

Finally, to add non-linearity to the model, the result is passed through two linear layers with ReLU activation and another layer normalization. The prediction is read with a linear layer with a sigmoid.

Overview of SAKT

Input:	Item sequence $1, \dots, t$ and interaction sequence $0, \dots, t-1$
Output:	Probability of answering the given item correctly
Mechanics:	Attention encoding of the sequences
Knowledge state:	Vector obtained from the attention encoding

In the original article, SAKT was tested on ASSISTments 2009, ASSISTments 2015, ASSISTments Challenge, Statics 2011, and a synthetic dataset. It achieved better AUC than DKT, DKT+, and DKVMN.

The hyper-parameters are an embedding dimension and a number of attention heads. The embedding dimension is the same for all elements in the model - items, interactions, and positions. The number of attention heads determines the number of parallel attention mechanisms that will contribute to the final attention encoding. Presumably, each head learns a different aspect of the input sequence.

4.5 Graph-based Knowledge Tracing (GKT)

The aim of GKT [25] is to take advantage of the graph structure of the knowledge concepts. So it reformulates knowledge tracing as a time series node-level classification problem - we have a graph of items and GKT in each time step predicts probabilities of answering each of them correctly. For authenticity purposes, I took the code of the model from `https://github.com/jhljx/GKT` and supplied it with own implementation of the knowledge graphs.

The overview of the prediction process can be found in figure 2.8 - it consists of aggregation, update and prediction phases. First, GKT concatenates hidden states of item nodes h_k^t with item embeddings into an intermediate representation h'^t_k - the items answered in time step t get $e^t E_e$ where e^t is the one-hot encoding of the interaction and E_e is the interaction embedding matrix; the neighbors of the answered item get $c^t E_c$ where c^t is one-hot encoded item and E_c is the item embedding matrix. That is the aggregation step and it can be summarized into the following assignment:

$$h'^t_k = \begin{cases} [h_k^t, e^t E_e] & k \text{ is the answered item} \\ [h_k^t, c^t E_c] & k \text{ is a neighbor of the answered item } i \end{cases} \tag{4.1}$$

During the update, the model calculates the next hidden state of the items h_k^{t+1}. The intermediate representation h'^t_k is passed through a multilayer perceptron (MLP) with two layers and

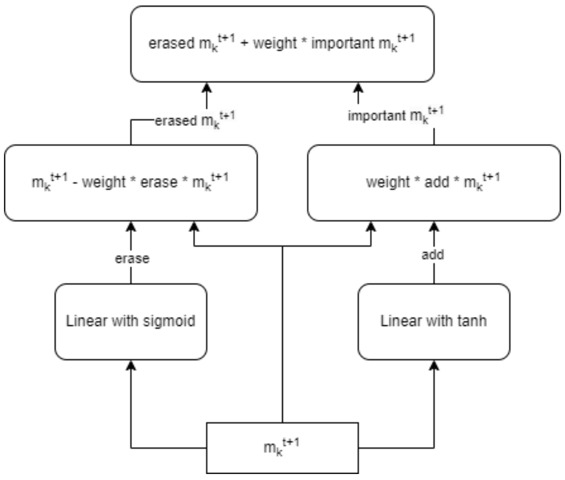


Figure 4.3 Erase-add gate scheme from extracted from GKT code

ReLU activation to incorporate the embeddings and the information about the newly answered item into the hidden states. There are two separate MLPs - one for processing the answered item f_{self} and one for the neighbors $f_{neighbor}$. Additionally, the neighbor MLP combines the intermediate representation of the neighbor with the answered item. Again, here is the equation summary:

$$m_k^{t+1} = \begin{cases} f_{self}(h'^t_k) & k \text{ is the answered item} \\ f_{neighbor}(h'^t_i, h'^t_k) & k \text{ is a neighbor of the answered item } i \end{cases} \tag{4.2}$$

The aggregated representations m_k^{t+1} are passed through an erase-add gate that presumably filters the important information from the representation. I created schema 4.3 from the published code. The output from the erase-add gate $\tilde{m}_k^{t+1}$ is passed into the GRU cell as input along with the previous hidden state. This operation is the core of the sequence prediction for every item node - whenever the item node is either answered or related to an answered item, the information is included in its hidden state. The next hidden state is computed followingly:

$$\tilde{m}_k^{t+1} = \mathcal{G}_{ea}(m_k^{t+1}) h_k^{t+1} = \mathcal{G}_{gru}(\tilde{m}_k^{t+1}, h_k^t) \tag{4.3}$$

Finally, GKT outputs probabilities for each item to be answered correctly in the next time step $t + 1$. They are calculated from the hidden states h_k^{t+1} with linear layer and sigmoid activation.

GKT leverages the graph structure of the items - however, this structure mostly isn't available in the knowledge tracing datasets. That's why the authors of GKT also studied the techniques how to obtain such structure. From their proposed methods, I chose and implemented three:

1. Dense adjacency - baseline all-to-all adjacency without self-loops, all edge weights equal to

$$\frac{1}{\text{num items - 1}}$$

2. Transitions adjacency - adjacency proportional to the number of times one item was answered after another (without self-loops), edge weight between item nodes i and j is equal to

$$\frac{n_{i,j}}{\sum_k n_{i,k}}$$

where $n_{i,j}$ is the number of times item j was answered immediately after item i

3. DKT-based adjacency - graph derived by the DKT-LSTM with a method proposed in [13] (section "Discovering Exercise Relationships"), the weight of an edge between items i and j is equal to
$$\frac{y(j|i)}{\sum_k y(j|k)}$$
where $y(j|i)$ is a probability predicted by DKT for item j after observing only a correct answer to item i, only when $i = j$ the edge weight is 0

Overview of GKT

Input:	Item sequence $1, \dots, t$ and interaction sequence $0, \dots, t-1$
Output:	Probability of answering the given item correctly
Mechanics:	Predictions are computed considering only the questioned item and its graph-neighbors which are also the only ones updated with new interaction
Knowledge state:	Graph's nodes with hidden state

GKT was originally tested on ASSISTments 2009 and Bridge to Algebra datasets. It overcame DKT and DKVMN in the prediction AUC. Additionally, the authors assessed GKT's performance using different graph structures, finding minimal differences in AUC among them.

GKT has three hyper-parameters: embedding dimension of items and interactions, hidden dimension of the graph nodes and a type of graph that is used for representation of relationships between items.

Apart from predictions, the model returns a list of hidden states representing the student's evolving knowledge state. The graph of items is crucial for the model's explainability and can be obtained directly from the GKT instance.

4.6 SAINT

SAINT [20] is an attention-based KT model that integrates information about both the question and the associated KC. This stands in contrast to other implemented models, which process only one type of item. In comparison with the SAKT model, another attention-based approach, SAINT distinguishes itself by employing multiple distinct attention mechanisms on top of each other to extract high-level information from the input. Furthermore, it leverages encoded exercise information as keys and values, while the correctnesses of student responses serve as queries (whilst SAKT utilizes interactions as values and keys and questions as queries).

The architecture of SAINT is depicted in figure 2.7, consisting of two main components: an encoder and a decoder. The implementation of SAINT was obtained from the published repository[2] and adapted for the use in this thesis.

As input, the model accepts exercise sequence $e_1, \dots, e_t$, comprising question IDs and corresponding KC IDs, alongside response sequence $S, a_1, \dots, a_{t-1}$, with a starting token S, denoting the student's answer correctness. The encoder embedding is constructed by aggregating question, KC, and position embeddings. On the other hand, the decoder embedding incorporates position embeddings with response embeddings, which range from binary (for EdNet and Assistments datasets) to ordinal categorical values (0, 0.25, 0.5, 0.75, 1) for the Marast dataset. This is different from the original article where the authors experimented only with binary answers.

The encoder module employs multi-head self-attention, followed by a feed-forward neural network (with 2 layers and ReLU activation) to process encoder embeddings into encoder output. Concurrently, the decoder begins by applying multi-head self-attention to decoder embeddings, generating a vector sequence that serves as a query for the final attention mechanism. This mechanism integrates the decoder output with the encoder output, acting as both a key and a

[2]`https://github.com/shivanandmn/SAINT_plus-Knowledge-Tracing-/tree/main`

value. The decoder process concludes with a feed-forward network, adding non-linearity to the decoding stage. These encoder and decoder blocks are stacked to extract high-level information from input sequences. Finally, the outcomes are derived via a linear layer with sigmoid activation.

Overview of SAINT

Input:	Question sequence and KC sequence $0, \ldots, t$ for the encoder and correctness sequence $0, \ldots, t-1$ for decoder
Output:	Probability of answering the given question correctly
Mechanics:	Attention-encode questions, attention-encode answer correctness and combine them with another attention mechanism
Knowledge state:	Decoder output vector (which is used for prediction)

SAINT introduces three key hyperparameters: embedding dimension, the number of attention heads, and the count of encoder/decoder stacks. Originally, it was compared with other variations of attention-based architectures and proved to be the best option.

4.7 Baselines

To test whether it makes sense to develop and maintain a complex neural network model, or whether a simpler model would be sufficient, I am going to compare the KT models with two baselines - LastAnswer and MeanAnswer.

LastAnswer

LastAnswer is inspired by the mentioned characteristics of the Assistments 2009 dataset where there is a higher probability that the correctness of the answer will remain the same in two consecutive time steps, than that it will change.

So the model LastAnswer takes in a sequence of answers $a_0, \ldots a_{t-1}$ and directly outputs it as a prediction for time steps $1, \ldots t$.

MeanAnswer

We want KT models to learn the underlying patterns of students' learning that we wouldn't be able to capture with simple statistics. Such simple statistics can be an average success rate on individual items which is what the MeanAnswer model predicts.

It calculates the mean correctness of each item during training - a number of times the item was answered correctly in the training dataset divided by the number of times the item occurred in the interactions. The result is the estimate of the difficulty of the item.

Chapter 5
Analysis

The objective of this thesis is to explore the field of knowledge tracing and propose its application in the novel educational platform. To achieve this, I have implemented a model from each category of KT approaches - basic sequence prediction, memory networks, attention mechanism, and graph-based model. The intention is to assess their predictive capabilities, compare their prediction patterns, and evaluate their interpretability. To the best of my knowledge, such a comprehensive analysis of KT models has not been created before. Previous studies have primarily focused on enhancing the prediction accuracy of the KT models or providing a theoretical overview of the field through surveys. However, little attention has been given to the predictions of the models and their practical applications.

5.1 Methodology

In this section, I describe the methodology adopted for the experimentation phase of this thesis. The steps include data partitioning, metric selection, training procedure, hyper-parameter optimization, and model assessment.

To ensure a comprehensive exploration of the field, a representative deep KT model has been chosen from each category, with implementation carried through either adaptation of existing published code or own development while adhering closely to the original article. For experimentation purposes, I have chosen ASSISTments 2009 and EdNet datasets due to their widespread utilization for benchmarking within the field. Additionally, data extracted from mathematics quizzes from the Marast system has been transformed into a novel KT dataset, enriched with textual features and featuring non-binary correctness evaluations. As evaluation metrics I have selected accuracy (ACC) and area under the receiver operating characteristic curve (AUC), following the general practices in the knowledge tracing domain.

I have separated 20% test subset from every dataset for final model comparison, while I split the remaining data into training, validation, and model-testing subsets. During the training, model parameters were iteratively optimized using the training data. The training was stopped either after a specified number of epochs or when the model failed to improve its metrics on the validation set for three consecutive iterations, invoking an early stopping criterion. Each model configuration underwent multiple training iterations, typically five (or less for computationally intensive models). At the end of each training iteration was the model measured on the test-model data subset. The model with the highest ACC was preserved for subsequent analysis.

To facilitate robust hyper-parameter comparison, metrics on model-test subsets from individual training runs were averaged before use. The decision to repeat runs was motivated by the necessity to stabilize the results, considering the minimal differences observed between configurations compared to the differences observed across runs. In instances where mean metrics were equivalent, preference was given to configurations resulting in fewer model parameters.

Upon identification of the optimal hyper-parameters, the best-performing model variant from multiple training iterations was utilized to evaluate ACC and AUC on the test set of each dataset, experimenting with both knowledge concepts and questions as items. Subsequently, the best-performing variant (questions or KCs) of each model was selected for each dataset to conduct the final comparison including models leveraging additional information.

5.2 Experiments

5.2.1 Prediction performance comparison

The metrics on the test subsets from the performance assessment on both types of items are depicted in figure 5.1. Notably, each dataset showcases a distinct optimal model, consistently with the ”no silver bullet” theorem in machine learning. Despite their diverse underlying mechanisms - ranging from memory to attention mechanisms - the KT models demonstrate remarkably similar prediction capabilities within a given dataset. These results are further illustrated in figure 5.2, where the probability predictions of the models are similar for a randomly selected student, indicating learning of shared underlying patterns.

The fundamental strategy of the KT models entails learning an item’s inherent difficulty, approximating it closely to its mean success rate (the probability the MeanAnswer baseline predicts). Subsequently, upon student interaction, the models adjust the probability accordingly: increasing it following the correct answer and decreasing it following an incorrect one. Variances among models primarily lie in their initial probability estimate for the previously unseen item and the magnitude of adjustment post-interaction. This trend is particularly evident in the Assistments dataset with KCs as items, where students commonly exercise a single KC for several consecutive interactions (the first plot from the top in figure 5.2).

The minimal differences observed in the predictions and their metrics suggest that the models may have reached the peak of performance achievable with the available information - a plain interaction sequence comprising item IDs and their correctness. Remarkably, even the simplest deep knowledge tracing model, DKT-RNN, achieved comparable performance to its counterparts under these conditions which should spark a debate about whether to use the more complex KT models in real-world scenarios (until the next generation of KT models with significantly better performance arrives).

Furthermore, it becomes apparent that considering questions as items (when working with datasets with a sufficiently high number of interactions per question) yields an advantage to the prediction performance. The reason is that the questions carry more granular information regarding the task the student encounters. It implies that KCs often encompass tasks with a spectrum of difficulty levels. This trait of the KT datasets is demonstrated by the MeanAnswer baseline that achieves higher ACC and AUC on questions as items compared to KCs as items on the same dataset.

5.2.2 Number of interactions per item

The table 5.1 gives into context the number of interactions per item, the difference in predictions of the KT models, and the best performance achieved on each of the datasets. The number of interactions per item is presented in quantiles of value counts - a higher number indicates that the models receive more comprehensive information about the item and can learn from a broader range of contexts. For instance, the 25% quantile for the Assistments KC is 234.0, indicating that only 25% of the KCs have fewer than 234 interactions.

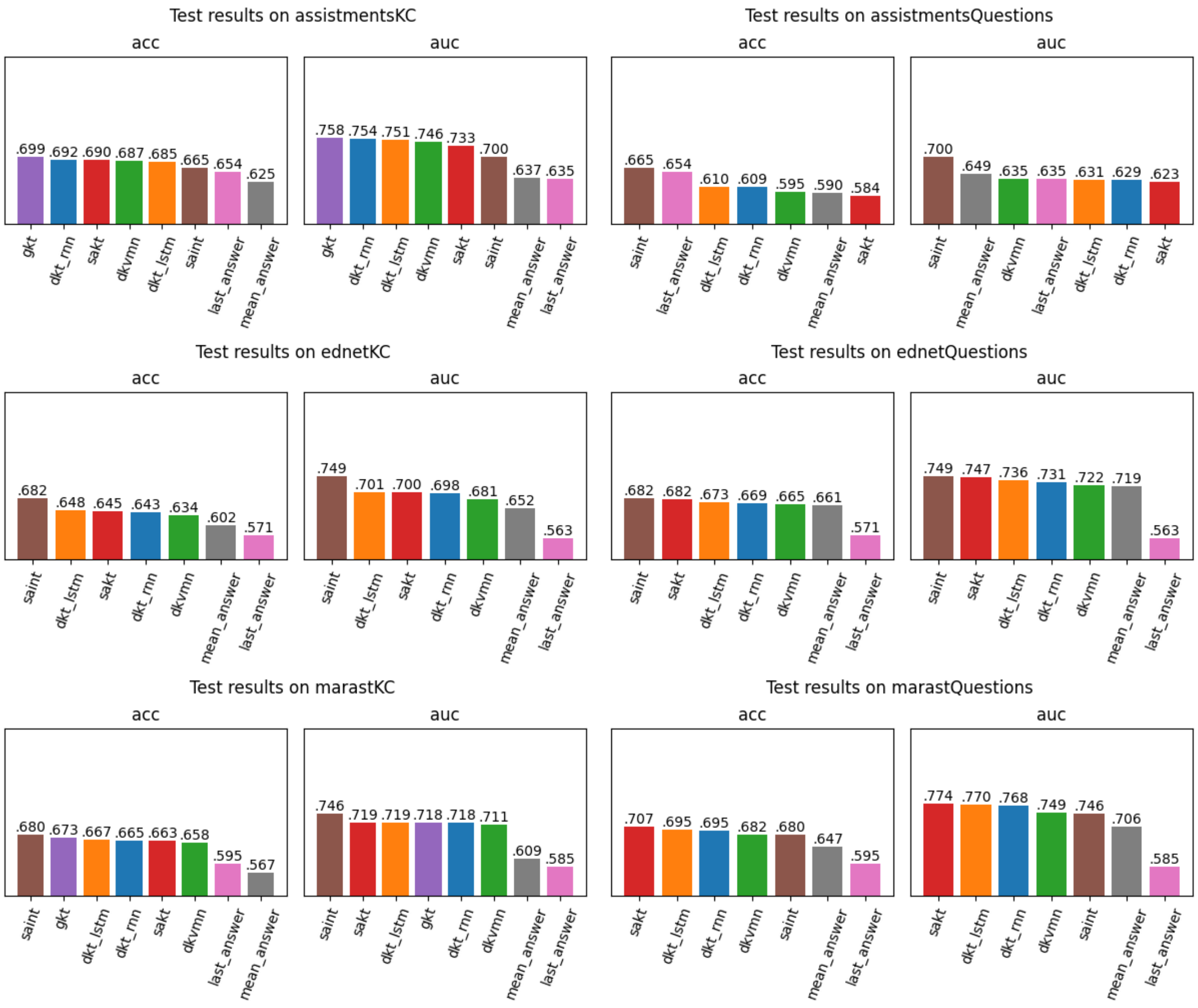


Figure 5.1 Performance of the KT models and baselines on the test subsets of the datasets.

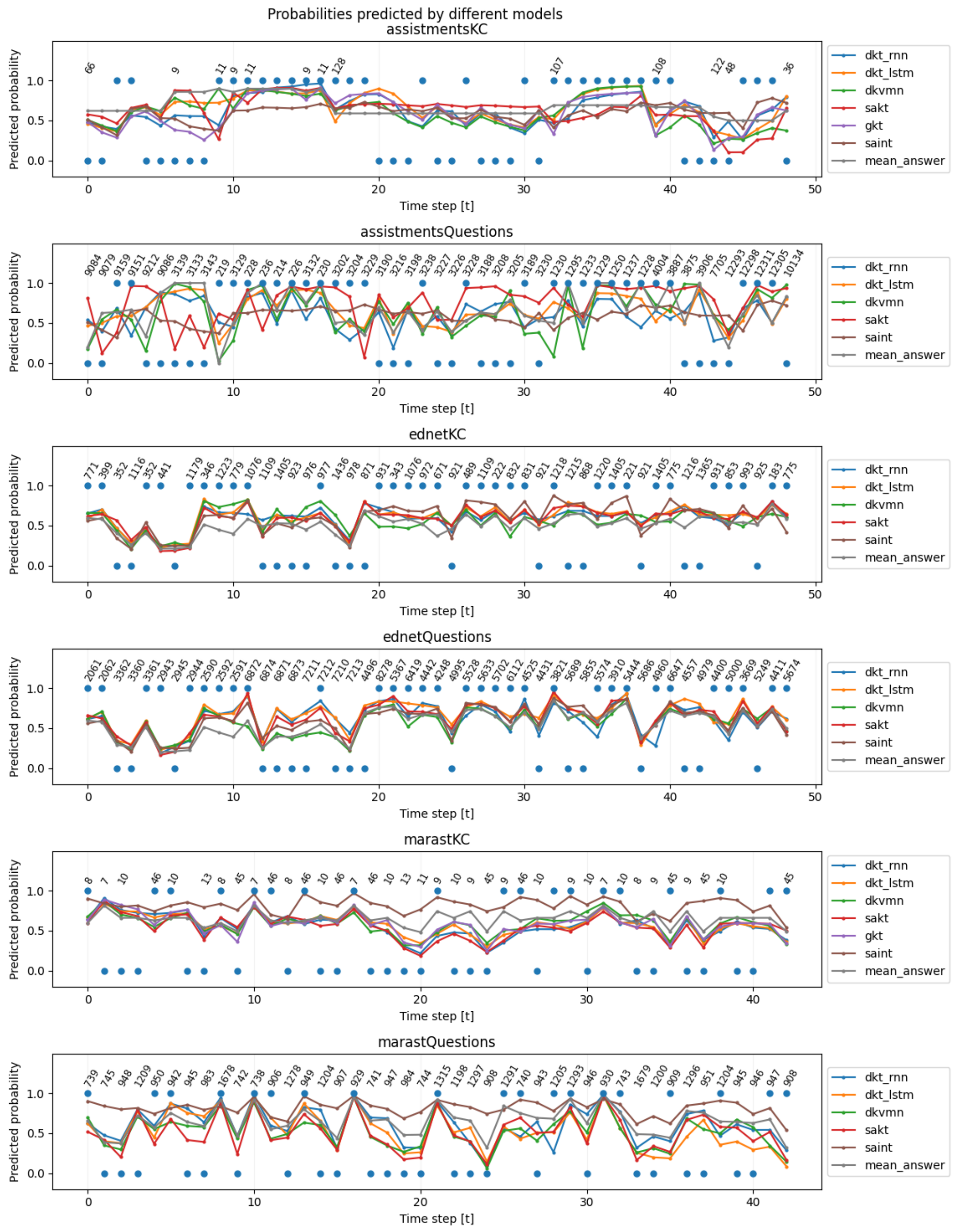


Figure 5.2 Comparison of predicted probabilities for one student from each dataset (the same student for questions and KCs dataset modes). Numbers displayed at the top of each plot indicate the IDs of answered items, while the circles denote the actual correctness of the answers (a circle at the top signifies a correct answer, and a circle at the signifies an incorrect answer).

	Assistments KC	Assistments Questions	EdNet KC	EdNet Questions	Marast KC	Marast Questions
25%	234.0	3.0	2794.0	533.0	1302.0	216.2
50%	906.0	7.0	5724.0	1425.0	2921.5	628.5
75%	1970.0	16.0	13630.5	2575.7	5025.2	1105.7
Avg diff	0.19	0.34	0.14	0.18	0.12	0.17
Best ACC	0.699	0.610	0.648	0.682	0.673	0.707
Best AUC	0.758	0.635	0.701	0.747	0.719	0.774

Table 5.1 Quantiles of number of interactions per item for all datasets, along with the average differences in predicted probabilities of the deep KT models

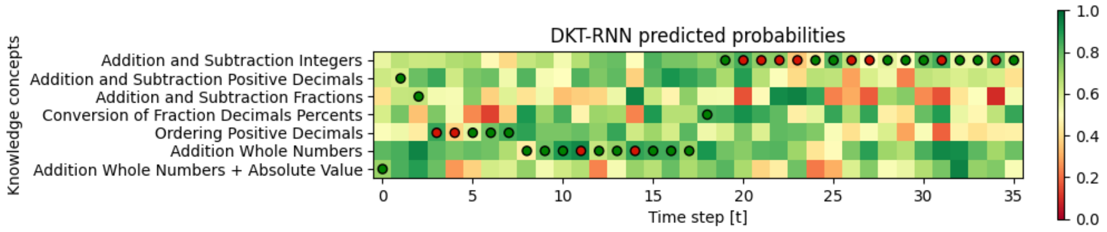


Figure 5.3 Visualization from DKT-RNN at Assistments KC showing predicted probabilities of all KC the student interacted with at some point of their sequence. The predicted probability is signified by the color, while the cirles represent which KC was answered (the row index) and how successfully (red signifies an incorrect answer, green a correct one).

Additionally, table 5.1 details the average difference of predictions of the KT models (excluding baselines). The Assistments Questions dataset, having the number of questions significantly large compared to the number of interactions, demonstrates substantial prediction variances and lower performance metrics across all KT models, indicating inadequate learning.

Moreover, when a dataset contains too few interactions per item, baseline models tend to outperform KT models. The baseline models, such as LastAnswer and MeanAnswer, leverage simple patterns within the dataset. We expect KT models to outperform these baselines as they utilize significantly more powerful modeling techniques and therefore consume more resources, like computational time and memory. Consequently, KT models outperforming the baselines is an indicator for gathering data (while building a new KT dataset) of reaching a sufficient amount of interactions.

It is crucial to note that the data remain significantly sparse, even with an adequate number of interactions per item. This means that a typical student interacts with only a small subset of available questions. Consequently, models capable of generalizing from seen items to unseen ones have a distinct advantage.

5.2.3 Analysis of the Models

The models process interaction sequences as inputs, including their correctness values. They either predict the probabilities for all items in the next step or take a query item as an additional input and only predict the probability for that item. The probabilities of the items the student interacted with can be visualized as shown in figure 5.2. For models that predict probabilities across all items, we can represent these visually as shown in figure 5.3. Additionally, models can provide supplementary visualization from their special mechanics (attention, memory, graphs, etc.).

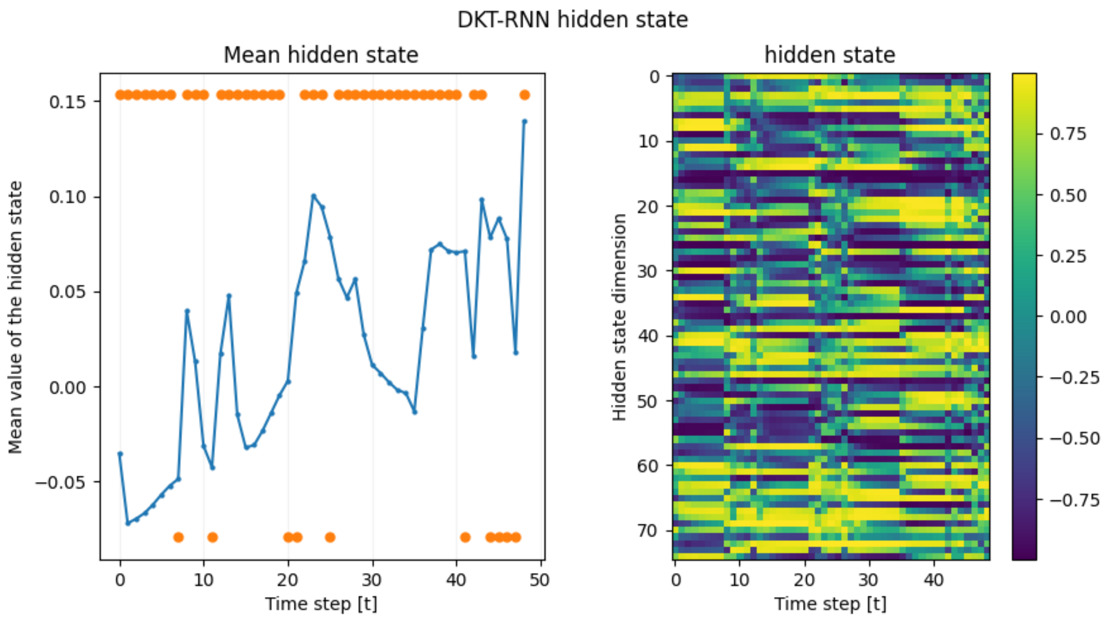


Figure 5.4 Progress of the hidden state of DKT-RNN during prediction. On the left, is a mean of the hidden state vector in each time step, the circles represent the actual correctness of the student's answer.

DKT

The DKT model processes sequences of encoded interactions and predicts the probability for each item at the next time step. It models the knowledge state using a single vector - the hidden state of a recurrent neural network. However, interpreting this state is challenging as its values frequently undergo significant changes through time. As depicted in figure 5.4, an element within the hidden state can completely alter its value in a single time step, as indicated by a change in color in one row. Also, using the average value of this state as an indicator of a student's understanding is not effective due to abrupt, non-intuitive shifts - a consecutive decrease even after a row of correct answers. Despite these challenges in interpretation, DKT maintains performance similar to other, more complex KT models, confirming that its internal mechanisms, while not transparent, are effective.

In figure 5.3 we see a visualization of the predicted probabilities for KCs in the Assistments dataset, limited to those interacted with by the student. An answer to one KC can significantly impact the predicted probabilities for another, which could be attributed to a student mastering prerequisites of the influenced KC or failing a related concept. This influence is not always intuitive - for instance, an initial correct answer in ”Addition and Subtraction Integers” reduces the probability for ”Addition and Subtraction Fractions”, but subsequent incorrect answers increase it.

DKVMN

DKVMN utilizes a key-value memory architecture to segment the knowledge state into latent KCs, maintaining their state according to the observed interactions. Through tuning of hyper-parameters, I have identified the optimal number of memory slots - five slots for all datasets with KCs as items, fifteen for Marast with questions, and twenty-five for EdNet with questions.

Figure 5.5 illustrates the inner workings of DKVMN's memory on two random students from Assistments KC. The right side shows probability components predicted by individual latent

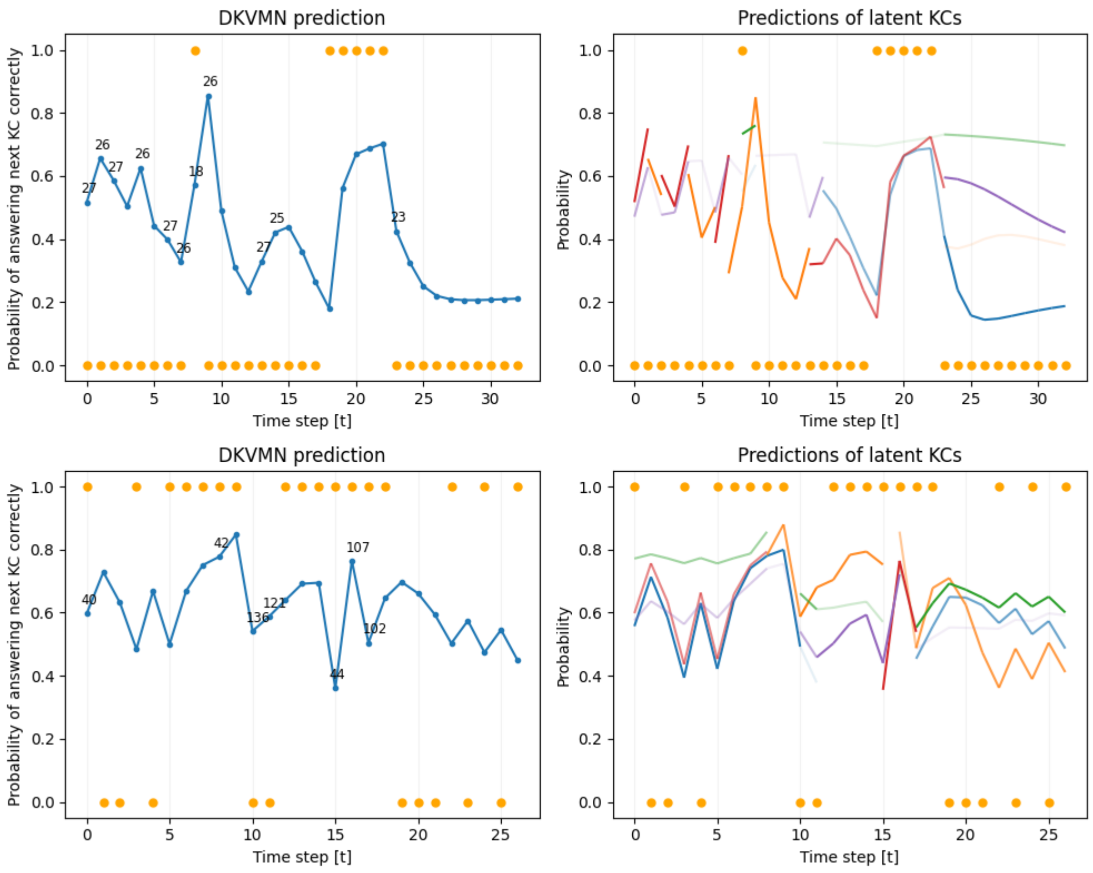


Figure 5.5 DKVMN predictions on Assistments with KCs (left) and the probability prediction from latent KCs with transparency set to the weights (amount of influence on the final prediction) (right)

KCs. Typically, only one or two latent KCs significantly determine the prediction. Furthermore, the latent KCs usually lead to similar predictions - the more memory slots the model has, the more redundancy can be seen in the plot. This is a common pattern for all datasets.

In contrast with DKT, DKVMN behaves more intuitively with a probability increase following a correct answer and a decrease after an incorrect one (observing sequences with multiple interactions with the same KC in a row). However, visualizations from EdNet questions with twenty-five memory slots revealed redundancy rather than diversity of the latent KCs' predictions, with all predicting similar probabilities.

SAKT

SAKT computes its predictions using the attention mechanism encoding past interactions, using the current question as a query. Figure 5.6 displays predictions for two students from the Assistments KC dataset, alongside with corresponding attention weights. These students practiced specific KCs and returned to them after engaging with other KCs. For instance, the first student revisited KC 25 at time step 16, and interestingly, SAKT paid significant attention to the interactions that a human would also mark as important - the last and second to last interaction with that KC and the last incorrect interaction with it. Consequently, SAKT decides to assign a high probability to the first new interaction with the KC 25.

SAKT exhibits the most significant variations in consecutive probabilities among all models. However, it also evaluates whether a change in the correctness of the answer was a guess or a slip or if the knowledge state of the student changed. Figure 5.6 offers illustrative examples: In the first scenario, the student slips and submits one incorrect answer amidst the correct ones for KC 100. Despite this, SAKT consistently predicts a high probability for this KC. Conversely, in the second scenario, despite two correct answers, the student struggles with understanding KC 86 at the sequence's end. SAKT evaluates the correct answers as guesses and maintains a low probability prediction for this KC. However, when the first student exercises KC 25 several times and then achieves the first correct response for that KC, SAKT promptly adjusts its predictions to a high probability for KC 25, correctly assessing that the student finally understood the KC and will answer it correctly in the future.

GKT

GKT is designed to leverage structural relationships among items by updating only the answered item and its neighboring items during prediction. I tested three types of graphs: a dense graph (a baseline graph that provides no additional structural information), a transition graph (with edges representing the number of times items were answered after each other), and a DKT graph (derived from predicted probabilities using DKT-LSTM). However, hyper-parameter tuning revealed minimal differences in GKT performance across these graph types, as depicted in figure 5.7.

This similar performance could suggest either an inability of GKT to effectively use the provided graph structure, or that the graphs do not adequately capture the underlying item relationships. The latter hypothesis is further explored in figure 5.8, where the Transition and DKT graphs are illustrated with KCs' descriptions from the Marast KC dataset. Each node displays its two strongest outgoing edges, and its color is determined by parent KCs which are defined through hierarchical numbering in the dataset. The width of each edge represents the strength of the relationship between the two KCs. The Transition graph naturally clusters related topics — an intuitively expected structure. However, its performance does not significantly surpass others. The DKT graph, conversely, is more dispersed with less pronounced differences in edge strengths.

Graphs for the Assistments dataset are even more dispersed, with no clear clusters forming for any graph type. Furthermore, the Assistments dataset lacks hierarchical structuring, making

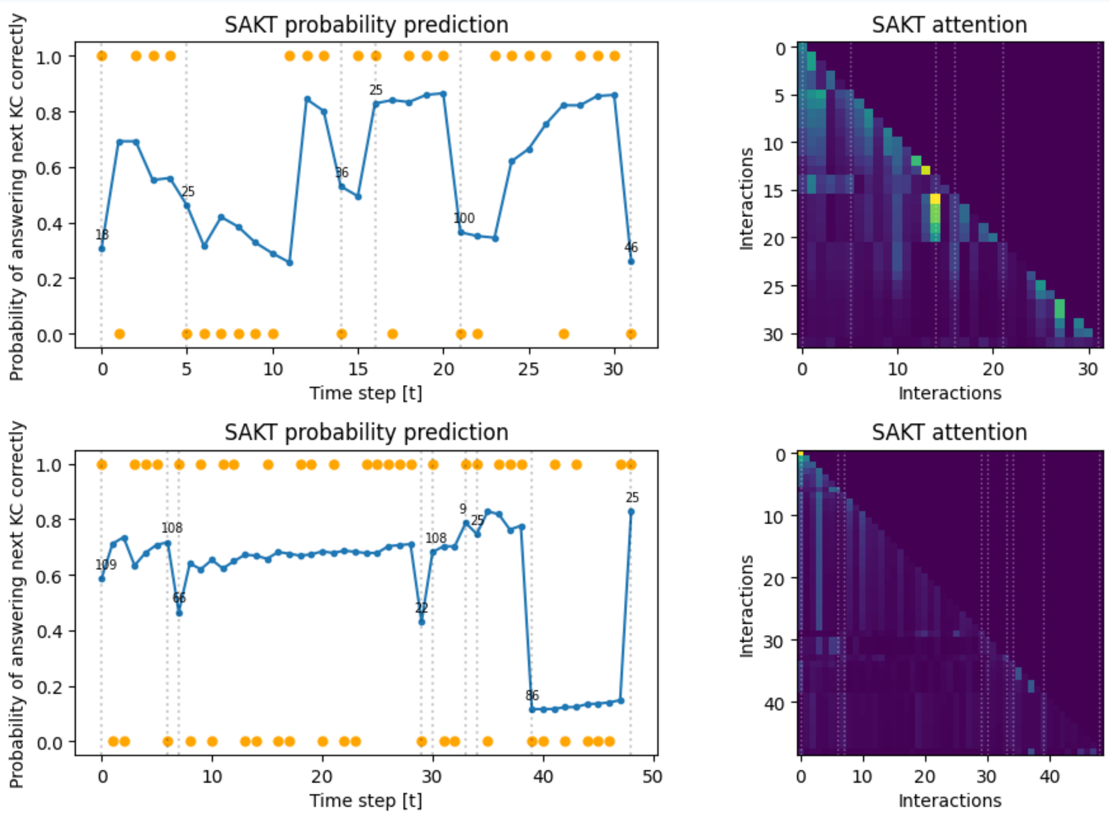
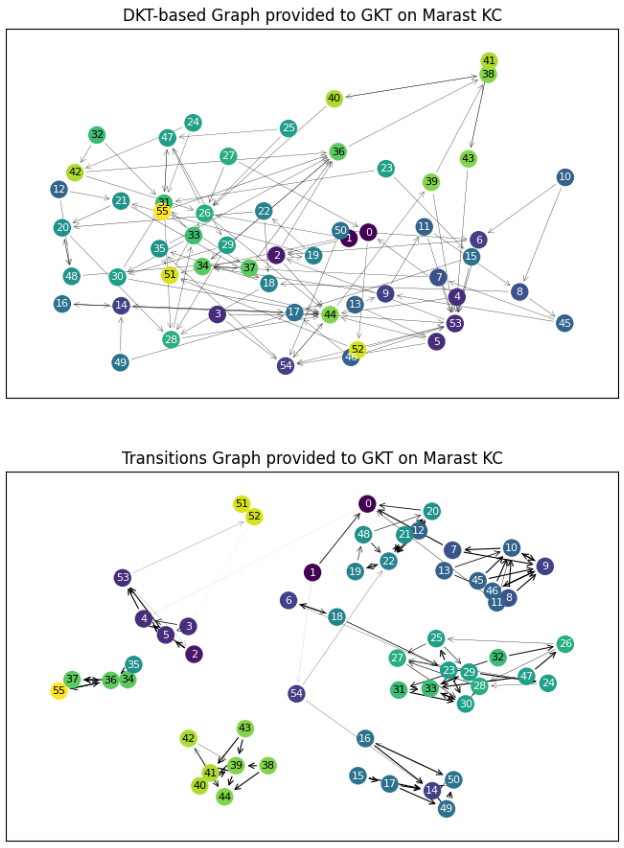


Figure 5.6 Predictions by SAKT on Assistments (left) with attention weights shown (right)

	Marast KC		Assistents KC	
	ACC	AUC	ACC	AUC
Dense graph	0.677	0.715	0.713	0.760
Transition graph	0.671	0.715	0.701	0.754
DKT graph	0.671	0.716	0.708	0.762

Figure 5.7 Performance of GKT with different types of graphs during hyper-parameter tuning


DKT-based Graph provided to GKT on Marast KC
Transitions Graph provided to GKT on Marast KC

0: Vlastnosti elementárních funkcí
1: Geometrická a aritmetická posloupnost
2: Funkce a zobrazení
3: Definiční obor, obor hodnot
4: Bijekce, injekce, surjekce
5: Monotonie
6: Posloupnosti
7: Vlastnosti posloupností
8: Vybraná posloupnost
9: Limita číselné posloupnosti
10: Definice limity
11: Přímý důkaz existence
12: Eulerovo číslo
13: Podílové kritérium
14: Řady
15: Nutná podmínka
16: d'Alembertovo kritérium
17: Srovnávací kritérium
18: Limita a spojitost funkce
19: Limita funkce
20: Výpočet
21: Vyvrácení existence
22: Spojitost funkce
23: Derivace
24: Výpočet derivace z definice
25: Výpočet derivace
26: Součty/součiny/podíly
27: Složená funkce
28: Inverzní funkce
29: Tečna grafu funkce
30: Průběh funkce
31: Asymptoty
32: Graf
33: Globální extrémy
34: Taylorovy polynomy
35: Výpočet
36: Mocninné řady
37: Odhad chyby
38: Integrální počet
39: Primitivní funkce
40: Výpočet pomocí per partes
41: Výpočet pomocí substituce
42: Parciální zlomky
43: Určitý integrál
44: Výpočet délky křivky
45: Sevřené posloupnosti
46: Hromadný bod posloupnosti
47: L'Hospitalovo pravidlo
48: Výpočet (s exponenciální funkcí, Eulerovým číslem)
49: Součet řady
50: Leibnizovo kritérium
51: Dodatky
52: Časté chyby
53: Vzor a obraz množiny
54: Landauova notace
55: Newtonova metoda

Figure 5.8 Graph of Marast KCs

evaluating the clustering abilities more difficult. Possibly, the fact that the Transition graph doesn't form clear clusters and at the same time limits GKT to updating only connected nodes, leads to suboptimal performance.

Interestingly, despite these findings, GKT performed best on the Assistments dataset using the DKT-based graph, suggesting that it may be the prediction mechanism rather than the graph utilization that drives GKT's superior performance on this dataset.

Nonetheless, the graph structure should theoretically provide valuable insights, particularly when item prerequisites are known, potentially enhancing prediction accuracy. However, the results of the experiments should warn us that the graph structure can also limit GKT's performance when designed poorly.

Unfortunately, GKT faced practical limitations, notably an ”Out of Memory” error when applied to datasets with a larger number of items (Marast with 2 382, EdNet with 13 523, and Assistments with 17 727 questions). This was caused by the model's graph representation and associated operations. Specifically, during the aggregation step, the model attempts to construct a matrix of size $2 \cdot |Y| \times 2 \cdot |Y|$ for computing node embeddings (for the whole graph at once), resulting in memory exhaustion. This limitation restricted further experimentation with the Assistments and Marast datasets, as other models achieved higher predictive performance with question-based items, providing richer item-specific information. Consequently, testing GKT with more detailed information became infeasible, possibly ruling out a good-performing model. Also, the error entirely prevented its application to the EdNet dataset.

SAINT

The objective of SAINT was to integrate the high-level insights from KCs with the more detailed information derived from individual questions. As indicated by the results in figure 5.1, SAINT successfully outperformed KC-based models on the EdNet and Marast datasets, demonstrating its ability to effectively incorporate the information from questions. However, its success in leveraging question-specific information appears limited on the Assistments dataset, likely due to the insufficient number of interactions per question. Consequently, other models that focus primarily on patterns within KCs tend to outperform SAINT on the Assistments dataset, capitalizing on the richer, more relevant data available from KC interactions.

SAINT emerges as the best-performing model among all KT models when evaluated on the EdNet dataset. This achievement could be attributed to its deep attention architecture, which enables it to process a substantially larger number of interactions and extract high-level patterns from it without over-fitting. Moreover, it is reasonable to assume that the model is sensitive to the relationship between questions and the assigned KC present in the utilized dataset. A dataset featuring more similar questions for each KC may prove more suitable for SAINT.

5.2.4 Textual features

The introduction of textual features in the Marast dataset presents a unique opportunity in the field of knowledge tracing, where such features are rare and typically limited to brief descriptions of knowledge concepts. Unfortunately, the absence of question texts has hindered the field's progress, as these texts can offer valuable insights for developing more advanced knowledge tracing models. During the limited time available with the Marast dataset, I experimented with integrating text information into existing knowledge tracing models.

In essence, this involved fine-tuning the BERT [39] (Bidirectional Encoder Representations from Transformers) NLP model and utilizing it to embed the question texts. These embeddings were then incorporated into the DKT-RNN and SAKT models, either independently or in conjunction with the original item embeddings.

Text embedding

In more detail, I leveraged the pre-trained BERT model [39] available from the Hugging Face model hub[1] — a platform for sharing pre-trained machine learning models. BERT, a transformer-based model, is designed for English language processing and trained using a masked language modeling (MLM) objective. In simple terms, during training, BERT receives a sentence with randomly masked words and attempts to predict these masked words, thereby learning bidirectional relationships within the sentence and producing a comprehensive embedding.

For this task, I selected the *google-bert/bert-base-cased* model, along with its corresponding tokenizer, which segments the sentence into smaller parts. Fine-tuning was performed on the training portion of the Marast dataset with the objective of predicting the correctness of answers for each question. Subsequently, I generated embeddings for all questions in the dataset, creating summary vectors that were stored for future use. This pre-computation of question embeddings facilitated faster processing during prediction and is valid because the text of the questions remains the same in the Marast system.

Text-enhanced DKT

As the first text-enhanced model, I implemented a variant of DKT-RNN referred to as DKT-text. This model incorporates text embeddings in place of the original interaction embeddings. In DKT-text, each question ID is assigned the BERT embedding and paired with its corresponding

[1] `https://huggingface.co/`

Model	ACC	AUC
DKT-text	0.677	0.737
DKT-combined	0.690	0.759
SAKT	0.698	0.767

Figure 5.9 Performance metrics of the text-enhanced models on the test subset of the Marast dataset

correctness value (for the Marast dataset, this is a categorical value from a list (0, 0.25, 0.5, 0.75, 1)). This concatenated input is then passed to the RNN cell for processing, following the same prediction procedure as the original DKT-RNN.

I conducted experiments to assess the efficiency of freezing the text embeddings versus allowing them to be trainable, utilizing BERT embeddings as the initial values. Results revealed that DKT-text with trainable embeddings achieved notably superior performance, with ACC of 0.668 and AUC of 0.729, compared to the 0.641 ACC and 0.695 AUC achieved with frozen embeddings (both values corresponding to the optimal hidden dimension).

Subsequently, I combined the text embeddings with the original question ID embeddings, leading to the creation of the DKT-combined model. During prediction, both the text embedding and the original lookup table embeddings are retrieved using the question ID. These embeddings are then concatenated with the correctness scalar and passed through a linear layer with ReLU activation. The resulting sequence of vectors, one for each time step, is then fed into the RNN cell for prediction.

Once more, the performance of the trainable question text embedding was notably superior, yielding an ACC of 0.682 and an AUC of 0.748, compared to the ACC of 0.645 and AUC of 0.696 attained with frozen text embeddings. Overall, DKT-combined surpassed DKT-text, as evidenced by the test performance summary table in Figure 5.9.

5.2.4.1 Text-enhanced SAKT

Lastly, I introduced a variant of the SAKT model aimed at testing a text-enhanced knowledge tracing approach with a query mechanism, in contrast to DKT, which solely leverages text for enhancing understanding of the student's learning history. This variant, which I will refer to as SAKT-text, utilizes BERT embeddings to initialize the question embedding. This embedding then transforms the question ID into an input for the linear transformation into an attention query. Essentially, the text of the question serves as a basis for selecting the relevant portion of the student's past interactions for prediction.

SAKT with trainable text embeddings achieved higher performance metrics of 0.696 ACC and 0.767 AUC compared to the untrainable-embedding variation with 0.651 ACC and 0.690 AUC. Notably, SAKT-text outperformed the text-enhanced DKT models while demonstrating performance similar to the previously presented KT models on the Marast dataset.

5.2.5 Results summary

The comprehensive results are depicted in figure 5.10. Several key observations emerge:

When sufficient interactions per question are available, the KT models exhibit enhanced performance when treating questions as items. In the opposite case, as in the Assistments dataset, it proves advantageous to treat KCs as items due to the richer information present in the dataset. Consequently, GKT gains a slight advantage on this dataset by leveraging its DKT-based graph to compensate for the limited question information. Conversely, SAINT struggles with the partial information about questions, resulting in poorer performance. Additionally, a notable characteristic of the Assistments dataset is the tendency of students to consecutively answer multiple questions related to the same KC. This sequential pattern presents another favorable scenario for GKT, which can focus on a relatively small subset of KCs consisting of the exercised KC and its direct neighbors.

EdNet stands out as the largest dataset in my experimentation, measured by the number of interactions. The high number of interactions favors the more complex models, such as SAINT and SAKT, which utilize either a higher parameter count or deeper architecture. However, EdNet proves to be the most challenging dataset for the models, as they generally exhibit lower performance on it, struggling to surpass the MeanAnswer baseline.

The Marast dataset presents a unique challenge to the KT models, both in terms of its features and structure. Remarkably, the simpler attention model, SAKT, achieves the highest performance. What sets SAKT apart from other KT models is its capability to select relevant interactions from the student's past and base the current prediction on them, making it the most suitable model for datasets featuring sets of switching items. In the Marast quiz system, students typically encounter a set of topics to practice or be examined on, with the same KC ID repeatedly recurring among other items within the set. SAKT demonstrates its ability to base predictions on the last occurrence of the current item rather than the last answered item, effectively considering the student's knowledge of it.

To explore the potential of incorporating full question texts into the modeling process, text-enhanced variants of DKT and SAKT were introduced for initial experimentation. Notably, a query item approach emerged as the most successful, leveraging the current question's text to query the student's past learning interactions. This sets an interesting direction for further research.

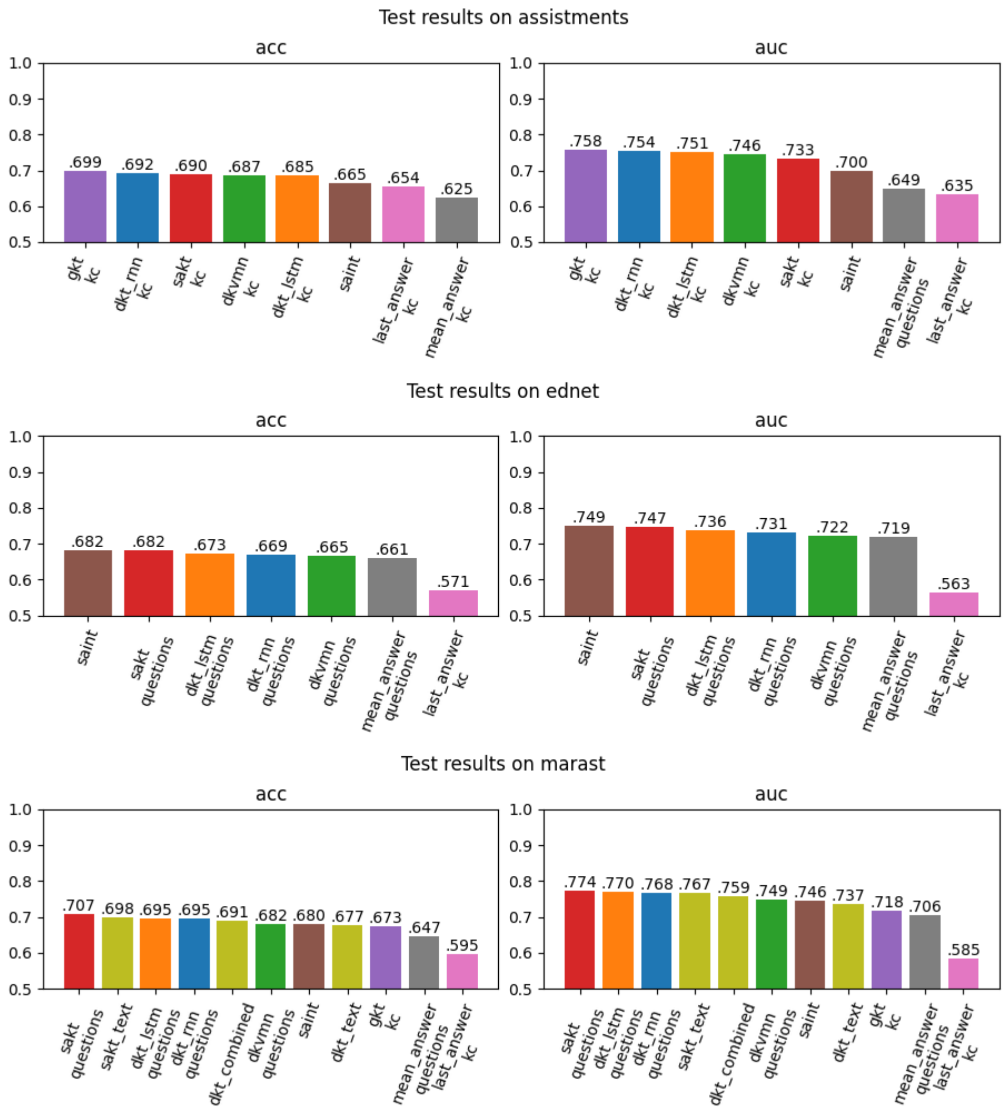


Figure 5.10 The best variants of all models on the datasets.

Chapter 6

Discussion

Knowledge tracing is a machine learning task with a rich history and a diverse range of models, encompassing various mechanisms aimed at enhancing prediction performance and explainability. These include attention mechanisms, memory-augmented neural networks, and graph neural networks.

Despite the diversity in mechanics, these models achieved comparable prediction AUC and ACC across commonly used datasets Assistments 2009 [31] and EdNet [37], and a newly created real-world dataset, Marast. However slight differences were observed and discussed. Notably, the graph-based model GKT [25] demonstrated its advantages on the Assistments dataset. An especially interesting feature was the ability to focus on one KC for multiple time steps (a recurrent pattern within this dataset). Additionally, GKT adeptly compensated for missing question information through its graph-based mechanics.

EdNet emerged as the most challenging dataset with generally the lowest prediction metrics. Experiments revealed that its extensive interaction data is best harnessed by more complex attention-based models SAINT [20] and SAKT [19], which derive predictions from the relevant pas interactions. However, despite various prediction mechanics and model complexities, the experiments led to similar prediction performance and similar prediction patterns. This suggests that current performance may already be optimized given the available sequence information, hinting at the need for more sophisticated approaches to further refine prediction accuracy. For instance, SAINT placed as the best-performing model for the EdNet dataset by combining high-level KC information with more granular question information.

This thesis introduces a novel KT dataset created from the data from FIT CTU's math quiz platform Marast. This dataset presents unique challenges and opportunities to the field of knowledge tracing. Namely, student sequences consist of sections, each with a distinct set of KCs being answered forming a pattern of items repeatedly reoccurring in time. This pattern was best leveraged by the attention-based model SAKT which learned to base its prediction on the last interaction with the same item in the past. Additionally, Marast dataset features KCs organized within a shallow hierarchy (mirroring the structure of the math course the quizzes belong to). Interestingly, the GKT model failed to fully exploit this hierarchical structure. This suggests the graph structure still has unused potential and may be a subject of future research.

The most significant opportunity presented by the Marast dataset lies in its textual features. They promise to provide rich semantical information for the knowledge state prediction which is an area that could not have been studied properly due to the lack of suitable datasets. This thesis takes the first steps toward leveraging the text features in this dataset. Two KT models with distinct mechanisms were adapted to utilize question text embeddings generated by the language model BERT, fine-tuned for the task at hand. Notably, employing question text as a query for prediction proved more beneficial than merely utilizing it to augment past interactions.

Practical applications

While knowledge tracing is in literature regarded as a critical task for personalized educational platforms, detailed implementation suggestions are notably absent. In my experience, knowledge tracing proves useful for immediate item probability prediction, facilitating personalized difficulty assessments and consequently more suitable exercise recommendations. This recommendation can be further explained by visualization of the KT model's underlying mechanics. For example, SAKT can directly point the student to the exercises that should be revised to improve the prediction probability of a desired task.

However, I have observed that knowledge tracing may not be suitable for estimating a student's long-term knowledge state and understanding, contrary to its original intention. The task's design, which accommodates sudden fluctuations in predicted probability and inner knowledge states, contradicts the gradual learning process inherent to human cognition.

Future directions

The consistent prediction performances and patterns observed suggest that knowledge tracing, as a pure sequence prediction task, has likely reached its pinnacle in terms of accuracy. As such, it is imperative for the research community to shift focus towards developing the next generation of models that leverage additional features and other information about the student's learning path. Of particular interest are the textual contents of questions, knowledge concepts, and student responses, which harbor rich semantic information capable of substantially enhancing performance. Consequently, the integration of text embeddings into KT models deserves the attention of KT researchers.

Another interesting future direction is long-term knowledge tracing. Here, the objective would shift from merely predicting the correctness of the next exercise answer to modeling the student's comprehension of the KCs over time. This paradigm shift necessitates a reevaluation and potential redefinition of the task at hand. One approach could involve optimizing predictions for the eventual correct/incorrect ratio of answers regarding a specific KC.

This thesis contributes to the efforts of the informatika.gg project, which is currently developing a novel educational application called Tiny. The findings of the experiments will guide the design of features and the data-gathering process.

Chapter 7

Conclusion

The aim of this thesis was to explore the field of knowledge tracing and its potential application within a personalized educational platform. Through an extensive literature review, I presented the fundamental approaches and mechanisms employed by KT models. Emphasis was placed on the investigation of diverse representations of the student's knowledge state, the intuition behind the prediction process of these models, and visualization techniques that could lead to real-world applications. Additionally, I curated a comprehensive list of knowledge tracing datasets, outlining their primary characteristics and metrics, and subsequently narrowed down the selection to the two most frequently utilized datasets, Assistments 2009 and EdNet, for a detailed analysis and experimentation. Moreover, I created a new KT dataset from real-world data gathered from the FIT CTU's math quiz platform, Marast. This dataset offers a full range of textual features, a non-binary answer evaluation, and unique sequence patterns challenging the existing KT models.

Subsequently, I undertook the implementation or adaptation of six distinct knowledge tracing models, alongside two baseline models for comparative purposes. Additionally, I modified two of them to include the pre-trained question text embeddings, taking the first steps towards a new generation of KT models.

Following the fine-tuning of the models' hyper-parameters, I carefully evaluated and compared their prediction performances and visualization capabilities. Notably, no model significantly outperformed the others on all datasets leading to the conclusion that each dataset contains different patterns for which we need different modeling mechanics. However, models leveraging further information like a combination of a question with KC (SAINT), graph structure (GKT), or question text (SAKT-text) were consistently placed among top-performing models for each dataset.

Moreover, I developed visualizations aimed at enhancing the interpretability of the knowledge tracing models, offering deeper insights into their prediction processes. Through detailed analysis and discussion, I outlined the strengths and limitations of the implemented methods. Finally, several future research directions were proposed.

Bibliography

1. VILLANO, Michael; BLOOM, Charles. Probabilistic Student Modeling with Knowledge Space Theory. 1992, p. 16.
2. CORBETT, Albert T.; ANDERSON, John R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*. 1994, vol. 4, pp. 253–278. Available also from: `https://doi.org/10.1007/BF01099821`.
3. GONG, Yue; BECK, Joseph E.; HEFFERNAN, Neil T. Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In: *Proceedings of the 10th International Conference on Intelligent Tutoring Systems - Volume Part I*. Pittsburgh, PA: Springer-Verlag, 2010, pp. 35–44. ITS'10. ISBN 3642133878. Available from DOI: `10.1007/978-3-642-13388-6_8`.
4. YUDELSON, Michael; KOEDINGER, Kenneth; GORDON, Geoffrey. Individualized Bayesian Knowledge Tracing Models. In: 2013, vol. 7926. ISBN 978-3-642-39111-8. Available from DOI: `10.1007/978-3-642-39112-5_18`.
5. BARRIA-PINEDA, Jordan; BRUSILOVSKY, Peter. Explaining Educational Recommendations through a Concept-Level Knowledge Visualization. In: *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion*. Marina del Ray, California: Association for Computing Machinery, 2019, pp. 103–104. IUI '19. ISBN 9781450366731. Available from DOI: `10.1145/3308557.3308690`.
6. CHAN, Ka Ian; TSE, Rita; LEI, Philip I.S. Tracing Students' Learning Performance on Multiple Skills Using Bayesian Methods. In: *Proceedings of the 6th International Conference on Education and Multimedia Technology*. Guangzhou, China: Association for Computing Machinery, 2022, pp. 84–89. ICEMT '22. ISBN 9781450396455. Available from DOI: `10.1145/3551708.3556202`.
7. RASCH, Georg. On General Laws and the Meaning of Measurement in Psychology. In: 1961. Available also from: `https://api.semanticscholar.org/CorpusID:5577238`.
8. WILSON, Kevin; KARKLIN, Yan; HAN, Bojian; EKANADHAM, Chaitanya. Back to the Basics: Bayesian extensions of IRT outperform neural networks for proficiency estimation. 2016.
9. CEN, Hao; KOEDINGER, Kenneth; JUNKER, Brian. Comparing Two IRT Models for Conjunctive Skills. In: 2008, pp. 796–798. ISBN 978-3-540-69130-3. Available from DOI: `10.1007/978-3-540-69132-7_111`.
10. PAVLIK JR, Phil; CEN, Hao; KOEDINGER, Kenneth. Performance Factors Analysis - A New Alternative to Knowledge Tracing. In: 2009, vol. 200, pp. 531–538. Available from DOI: `10.3233/978-1-60750-028-5-531`.

11. VIE, Jill-Jênn; KASHIMA, Hisashi. Knowledge Tracing Machines: Factorization Machines for Knowledge Tracing. *CoRR*. 2018, vol. abs/1811.03388. Available from arXiv: `1811.03388`.

12. GAN, Wenbin; SUN, Yuan; PENG, Xian; SUN, Yi. Modeling learner's dynamic knowledge construction procedure and cognitive item difficulty for knowledge tracing. *Applied Intelligence*. 2020. Available from DOI: `10.1007/s10489-020-01756-7`.

13. PIECH, Chris; SPENCER, Jonathan; HUANG, Jonathan; GANGULI, Surya; SAHAMI, Mehran; GUIBAS, Leonidas J.; SOHL-DICKSTEIN, Jascha. Deep Knowledge Tracing. *CoRR*. 2015, vol. abs/1506.05908. Available from arXiv: `1506.05908`.

14. YEUNG, Chun-Kit; YEUNG, Dit-Yan. Addressing Two Problems in Deep Knowledge Tracing via Prediction-Consistent Regularization. *CoRR*. 2018, vol. abs/1806.02180. Available from arXiv: `1806.02180`.

15. MINN, Sein; YU, Yi; DESMARAIS, Michel C.; ZHU, Feida; VIE, Jill-Jênn. Deep Knowledge Tracing and Dynamic Student Classification for Knowledge Tracing. In: *2018 IEEE International Conference on Data Mining (ICDM)*. 2018, pp. 1182–1187. Available from DOI: `10.1109/ICDM.2018.00156`.

16. ZHANG, Jiani; SHI, Xingjian; KING, Irwin; YEUNG, Dit-Yan. *Dynamic Key-Value Memory Networks for Knowledge Tracing*. 2017. Available from arXiv: `1611.08108 [cs.AI]`.

17. WANG, Chunpai; SAHEBI, Shaghayegh. Continuous Personalized Knowledge Tracing: Modeling Long-Term Learning in Online Environments. In: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. Birmingham, United Kingdom: Association for Computing Machinery, 2023, pp. 2616–2625. CIKM '23. ISBN 9798400701245. Available from DOI: `10.1145/3583780.3614822`.

18. HE, Liangliang; LI, Xiao; WANG, Pancheng; TANG, Jintao; WANG, Ting. MAN: Memory-Augmented Attentive Networks for Deep Learning-Based Knowledge Tracing. *ACM Trans. Inf. Syst.* 2023, vol. 42, no. 1. ISSN 1046-8188. Available from DOI: `10.1145/3589340`.

19. PANDEY, Shalini; KARYPIS, George. *A Self-Attentive model for Knowledge Tracing*. 2019. Available from arXiv: `1907.06837 [cs.LG]`.

20. CHOI, Youngduck; LEE, Youngnam; CHO, Junghyun; BAEK, Jineon; KIM, Byungsoo; CHA, Yeongmin; SHIN, Dongmin; BAE, Chan; HEO, Jaewe. Towards an Appropriate Query, Key, and Value Computation for Knowledge Tracing. *CoRR*. 2020, vol. abs/2002.07033. Available from arXiv: `2002.07033`.

21. SHIN, Dongmin; SHIM, Yugeun; YU, Hangyeol; LEE, Seewoo; KIM, Byungsoo; CHOI, Youngduck. SAINT+: Integrating Temporal Features for EdNet Correctness Prediction. *CoRR*. 2020, vol. abs/2010.12042. Available from arXiv: `2010.12042`.

22. GHOSH, Aritra; HEFFERNAN, Neil; LAN, Andrew S. *Context-Aware Attentive Knowledge Tracing*. 2020. Available from arXiv: `2007.12324 [cs.LG]`.

23. ZHAO, Jinjin; BHATT, Shreyansh; THILLE, Candace; ZIMMARO, Dawn; GATTANI, Neelesh. Interpretable Personalized Knowledge Tracing and Next Learning Activity Recommendation. In: *Proceedings of the Seventh ACM Conference on Learning @ Scale*. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 325–328. L@S '20. ISBN 9781450379519. Available from DOI: `10.1145/3386527.3406739`.

24. ZHU, Mengxia; HAN, Siqi; YUAN, Peisen; LU, Xuesong. Enhancing Programming Knowledge Tracing by Interacting Programming Skills and Student Code. In: *LAK22: 12th International Learning Analytics and Knowledge Conference*. Online, USA: Association for Computing Machinery, 2022, pp. 438–443. LAK22. ISBN 9781450395731. Available from DOI: `10.1145/3506860.3506870`.

25. NAKAGAWA, Hiromi; IWASAWA, Yusuke; MATSUO, Yutaka. Graph-based Knowledge Tracing: Modeling Student Proficiency Using Graph Neural Network. In: 2019, pp. 156–163. ISBN 978-1-4503-6934-3. Available from DOI: 10.1145/3350546.3352513.

26. TONG, Shiwei; LIU, Qi; HUANG, Wei; HUNAG, Zhenya; CHEN, Enhong; LIU, Chuanren; MA, Haiping; WANG, Shijin. Structure-Based Knowledge Tracing: An Influence Propagation View. In: *2020 IEEE International Conference on Data Mining (ICDM)*. 2020, pp. 541–550. Available from DOI: 10.1109/ICDM50108.2020.00063.

27. YANG, Yang; SHEN, Jian; QU, Yanru; LIU, Yunfei; WANG, Kerong; ZHU, Yaoming; ZHANG, Weinan; YU, Yong. *GIKT: A Graph-based Interaction Model for Knowledge Tracing.* 2020. Available from arXiv: 2009.05991 [cs.AI].

28. TU, Liang; ZHU, Xinning; JI, Yang. Graph-Based Dynamic Interactive Knowledge Tracing. In: *Proceedings of the 2023 8th International Conference on Distance Education and Learning.* Beijing, China: Association for Computing Machinery, 2023, pp. 49–56. ICDEL '23. ISBN 9798400700422. Available from DOI: 10.1145/3606094.3606124.

29. LU, Yu; CHEN, Penghe; PIAN, Yang; ZHENG, Vincent W. CMKT: Concept Map Driven Knowledge Tracing. *IEEE Transactions on Learning Technologies.* 2022, vol. 15, no. 4, pp. 467–480. Available from DOI: 10.1109/TLT.2022.3196355.

30. LIU, Dong; ZHANG, Yunping; ZHANG, Jun; LI, Qinpeng; ZHANG, Congpin; YIN, Yu. Multiple Features Fusion Attention Mechanism Enhanced Deep Knowledge Tracing for Student Performance Prediction. *IEEE Access.* 2020, vol. 8, pp. 194894–194903. Available from DOI: 10.1109/ACCESS.2020.3033200.

31. FENG, Mingyu; HEFFERNAN, Neil T.; KOEDINGER, K. Addressing the assessment challenge with an online system that tutors as it assesses. *User Modeling and User-Adapted Interaction.* 2009, vol. 19, pp. 243–266. Available also from: https://api.semanticscholar.org/CorpusID:13028204.

32. XIAO, Yongkang; XIAO, Rong; HUANG, Ning; HU, Yixin; LI, Huan; SUN, Bo. Knowledge Tracing Based on Multi-Feature Fusion. *Neural Comput. Appl.* 2022, vol. 35, no. 2, pp. 1819–1833. ISSN 0941-0643. Available from DOI: 10.1007/s00521-022-07834-w.

33. WANG, Zhifeng; HOU, Y.; ZENG, Chunyan; ZHANG, Sixv; YE, Ruiqiu. Multiple Learning Features–Enhanced Knowledge Tracing Based on Learner–Resource Response Channels. *Sustainability.* 2023. Available also from: https://api.semanticscholar.org/CorpusID:259422239.

34. ABDELRAHMAN, Ghodai; WANG, Qing; NUNES, Bernardo. Knowledge Tracing: A Survey. *ACM Comput. Surv.* 2023, vol. 55, no. 11. ISSN 0360-0300. Available from DOI: 10.1145/3569576.

35. WANG, Yutao; HEFFERNAN, Neil T.; HEFFERNAN, Cristina. Towards better affect detectors: effect of missing skills, class features and common wrong answers. In: *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge.* Poughkeepsie, New York: Association for Computing Machinery, 2015, pp. 31–35. LAK '15. ISBN 9781450334174. Available from DOI: 10.1145/2723576.2723618.

36. BAKER, Ryan; GOWDA, Sujith; CORBETT, Albert. Towards Predicting Future Transfer of Learning. In: 2011, pp. 23–30. ISBN 978-3-642-21868-2. Available from DOI: 10.1007/978-3-642-21869-9_6.

37. CHOI, Youngduck; LEE, Youngnam; SHIN, Dongmin; CHO, Junghyun; PARK, Seoyon; LEE, Seewoo; BAEK, Jineon; KIM, Byungsoo; JANG, Youngjun. EdNet: A Large-Scale Hierarchical Dataset in Education. *CoRR.* 2019, vol. abs/1912.03072. Available from arXiv: 1912.03072.

38. ZHANG, Aston; LIPTON, Zachary C.; LI, Mu; SMOLA, Alexander J. *Dive into Deep Learning.* Cambridge University Press, 2023. https://D2L.ai.

39. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. 2018, vol. abs/1810.04805. Available from arXiv: `1810.04805`.

Contents of the attachment

```
readme.txt ........... brief description of the contents of the attachment
src
  impl ............... source codes of the thesis implementation
  thesis ............. source code for thesis text in LaTeX
text ................. text of the thesis
  thesis.pdf ......... text of the thesis in PDF format
```