Applying Knowledge Tracing to Predict Exercise Response Time

Shamiran Jaf
Faculty of Engineering
Lund University
Lund, Sweden
shamiran.jaf@gmail.com

Sepehr Noorzadeh
Faculty of Engineering
Lund University
Lund, Sweden
sepehr.noorzadeh@gmail.com

***Abstract*—*Deep knowledge tracing* is an emergent field of study which applies neural networks to the task of measuring student knowledge on an interaction-to-interaction basis. It involves using an interaction history to predict some aspect of future interactions in an educational context. However, all research in this topic has been focused on predicting the *correctness* of future exercises. This paper aims to adapt existing knowledge tracing models to also predict the *response time* of future exercises. To the best of our knowledge, this has not been attempted before. We found that using some adjustments to existing models, response time prediction is possible. We believe that response time prediction can be a great tool for calculating the fluency of students since fluency is strongly dependent on the speed at which students solve exercises.**

***Index Terms*—machine learning, educational technology, data mining**

I. Introduction

Education is an important foundation for equality and societal prosperity, while teachers struggle to give each student the time and personalized learning they need. Learning is at the same time increasingly taking place on digital platforms, allowing for new opportunities to help teachers achieve the goal of providing an optimal learning experience for every student. The field of *knowledge tracing* [1] aims to capitalize on one of these opportunities: the new-found ability to track students interactions with curricula on an interaction-to-interaction basis.

Using online platforms such as *intelligent tutoring systems*, a wealth of new information can be gathered for each interaction with an exercise. That information includes whether the student has answered correctly or incorrectly and how much time they took to answer. Previous attempts at quantifying student knowledge on an interaction-to-interaction basis have used laboriously constructed algorithmic models [1]. A new branch of *knowledge tracing* has successfully applied different AI models to predict student performance, measured by the correctness of their response on exercises. These models don't need to be explicitly programmed, and have shown performance improvements over their algorithmic counterparts [2].

However, current models that only predict correctness are limited in their potential applications; making choices in how to adapt learning paths for a student's requires a deeper understanding of the student's fluency in the concepts that the learning path is composed of.

In this paper, we show that the models used for correctness prediction can be adapted for predicting response time. Response time can be used to determine *processing speed*, which is an important component of fluency in early mathematical education [3], [4].

In addition, we found that both response time and correctness prediction performance can be improved by adding handcrafted features.

To the best of our knowledge, this is the first work to apply *knowledge tracing* models to the task of response time prediction. To this end, we have managed to produce promising results, where our models have a greater predictive performance than a statistical baseline.

A. Task formulation

We denote a sequence of interactions between a student and exercises as $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, where the interaction $\mathbf{x}_t$ consists of information about the exercise being answered $\mathbf{e_t}$, along with the student's response to the previous exercise, denoted as $\mathbf{r_{t-1}}$. Previous knowledge tracing models have been concerned with predicting the correctness of the next response $\mathbf{r_t}$. The task we addressed in this paper, however, is to predict the response time – the time it takes for a student to respond to an exercise – for the response $\mathbf{r_t}$, as shown in Figure 1.

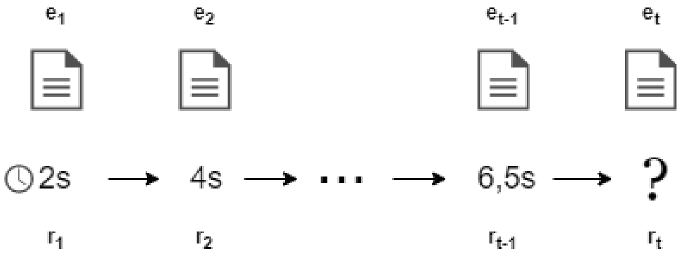


Fig. 1: A sequence of interactions where each response has a response time measured in seconds.

II. Related works

Knowledge tracing refers to the usage of statistical models to track the knowledge levels of students across different subjects or concepts. The first knowledge tracing models used Bayesian statistics and Markov chains to model student knowledge with promising initial results [1].

The goal of knowledge tracing is to, given a sequence of student interactions with a learning platform, predict some aspect of their future interactions. So far, the field of knowledge tracing has been focused on the task of predicting the correctness of a response, which can be either correct or incorrect. The related works below are thus all concerned with correctness.

A. Deep Knowledge Tracing

Introduced in [2], Deep Knowledge Tracing (DKT) refers to the application of machine learning models to the task of knowledge tracing. Inspired by the success of Recurrent Neural Networks (RNNs) [5] in language processing, Long-Short Term Memory (LSTM) [6] networks were applied to the task of correctness prediction.

The authors were able to achieve a considerable improvement in performance compared to Bayesian knowledge tracing models when predicting correctness. In contrast to Bayesian knowledge tracing [1] models, the DKT model – being a neural network – will itself learn the relationships between different knowledge concepts. Therefore, the DKT model does not require the same degree of expert knowledge in the subject matter.

Other works, such as [7] and [8], were successful in augmenting the DKT model through the usage of memory-augmented neural networks.

B. Self-Attentive Knowledge Tracing

Recurrent neural networks were augmented by a new mechanism called *attention* by [9]. Attention allowed neural networks to learn more long-term relations by allowing any position in a time series to directly relate to an earlier position.

Eventually, it was discovered by [10] that attention can be a powerful mechanism on its own without being applied to an RNN. The paper introduced the *multi-headed self-attention* layer which could model long term relationships between samples in time-series data.

Multi-headed attention was applied to knowledge tracing by [11] to create a new model called the *Self Attentive Knowledge Tracing* (SAKT).

The SAKT architecture showed an improvement in performance over earlier models. In addition, the SAKT model is much faster to train on average than the aforementioned models thanks to much higher training parallelism. This higher training parallelism is in turn enabled by the lack of recurrent architecture, and the usage of multiple independently trained attention heads.

C. Relation-Aware Knowledge Tracing

Pandey and Srivastava [12] augmented the SAKT model by modeling student forgetfulness and exercise relation. The forgetting behaviour was modelled by looking at the *timestamp* fields typically found in knowledge tracing datasets. The exercise relation modelling was made by comparing the text content of the exercises using *natural language processing* methods.

D. Separated Self-Attentive Neural Knowledge Tracing

Vaswani et al. [10] not only introduced the multi-head self-attention layer, but utilized it for natural language processing by arranging several layers of multi-head self-attention into *encoder* and *decoder* blocks.

Choi et al. [13] used a similar architecture to create a new knowledge tracing model called *Separated Self-Attentive Neural Knowledge Tracing* (SAINT) that outperformed the state of the art knowledge tracing models at the time.

The SAINT model was later augmented by [14] by incorporating the temporal features available in the EdNet dataset to create *SAINT+*. SAINT+ uses new time-related features such as *response time* and *lag time* to more accurately predict correctness for exercises in the *EdNet* dataset [15].

E. Last Query Transformer RNN

Jeon [16] introduced a novel sequence-to-sequence architecture as a winning entry in the Kaggle competition *Riiid AIEd Challenge 2020*. The architecture is noteworthy for combining an attention-based model with an RNN network. In addition, the model also utilizes an engineering trick to reduce the complexity of the matrix multiplications associated with training. This optimization incurs a slight accuracy loss, but also allows for the use of longer input sequences. The longer sequences can, depending on the dataset, offset the aforementioned loss of accuracy for a net gain in performance.

III. Datasets

In order to train models, labeled training data is needed. Online tutoring systems have allowed for the collection of such data, and several publicly available datasets gathered from tutoring systems have been used in this study. These datasets have been used in earlier knowledge tracing studies. In addition to these datasets, we also have access to a private dataset called the *Akribian* dataset.

Name	**Rows**	**Categories**	**Unique exercises**
Akribian	71,413	239	787
ASSISTments 2012	2,630,080	199	50,989
Junyi Academy	16,217,311	10	1,327
EdNet	101,230,332	7	13,525

TABLE I: Summary of datasets.

Every dataset is tabular and each row describes one interaction between a student and the tutoring system. Table I shows the number of rows, the number of categories into which exercises are grouped and the number of unique exercises for each dataset. The datasets are largely similar with regards to structure and the type information they contain.

A. Akribian dataset

The Akribian dataset is extracted from the game *Count on me!* which can be played on tablets. The purpose of the game is to teach 6-9 year old children basic maths.

The game's progression is linear and personalization mainly consists of skipping or repeating learning sequences. As such, there isn't much variance in the sequence of questions answered by each student.

B. ASSISTments dataset

The ASSISTments dataset consists of user interactions with exercises in ASSISTments, an online education tool described in [17]. It is an online system that allows teachers to assign math exercises to students who then can use the platform to complete exercises while receiving both automated hints and help from their teachers.

There are several versions of the ASSISTments dataset available publicly, and each dataset differs slightly with regards to the what type of data it contains. ASSISTments 2012 was chosen for this paper as it contains timestamps, which can be used to engineer better features. In addition, it is the largest of the ASSISTments datasets.

C. Junyi Academy dataset

The Junyi Academy dataset consists of user interactions with exercises in Junyi Academy, an online education tool derived from Khan Academy and mainly used in Taiwan. Junyi Academy, similar to ASSISTments, is an online system for assigning and assessing maths problems for students. The dataset is hierarchical with four levels of subdivision.

D. EdNet dataset

The EdNet dataset as described by [15] is a large-scale hierarchical dataset collected from *Santa*, a mobile app developed by *Riiid!* that tutors more than 780,000 South Korean students in English. It has recently become a popular dataset in benchmarking the performance of knowledge tracing models, largely thanks to a *Kaggle* competition where users submitted models that competed on the dataset.

IV. Approach

To solve the two tasks defined in the introduction, we have used several of the sequence-to-sequence models that have previously been applied for correctness prediction. The models are, in chronological order of introduction, DKT [2], SAKT [11], SAINT [13] and LQTR [16]. We first implemented the original correctness and benchmarked their performance in the correctness prediction task. We then modified each model to predict response time.

In addition, we used data processing to augment the datasets with additional features that can be used by the models to model more complex relations.

A. Data processing

The datasets used in this study differ in the quantity, types and format of their features. Therefore it was necessary to implement a pipeline for processing the datasets so that they all adhere to the same format and specifications. This allows every dataset to be used with every model without the need for modifications to the model.

1) Formatting the data: The first modification was using integer indices for exercise and category identifiers. Normally these fields would be represented by strings, representing either a human-readable title, or a unique identifier used internally by a database. Since strings are not able to be turned into embeddings for inputting into the model, these values were converted into integers. This was done by creating lists of unique exercise and category identifiers and converting the strings into integers representing their indices in these lists.

In addition, all timestamps were converted to the *seconds since Unix Epoch* format in order to make it possible to calculate time differences in seconds. Rows lacking data were filtered out, as well as rows containing response time values that were larger than the 90th percentile of response time values. The reason for this was to filter out outliers resulting from users leaving their computer devices without answering the exercise.

2) Sorting the datasets: Some of the datasets contain millions of entries and require several gigabytes of RAM capacity to load. As such, they could not be read in their entirety in one pass. To address this issue, we developed a framework to read the datasets a predetermined number of rows at a time. The interactions in the datasets are stored in chronological order of the responses, since that is the norm in most databases used in online services. This means that when reading the dataset in parts, there would be no guarantee that the entire interactions history of the users would be loaded into the memory at once. This could lead to the extraction of several disjoint user exercise histories that are actually fragments of the same user history. This issue could damage the model's performance since it wouldn't have access to the entire exercise interactions history of each user.

To solve this problem, we first sorted the database by user ID so that every user's interactions would be in a continuous order. Each user's interactions was then sorted by timestamp in an ascending order to make sure that the exercises are in chronological order.

3) Padding and windowing: After sorting, we cut the exercise history for each user into windows of a specific size. Exercise histories that were below the window size were padded to be the correct size, and those that were over the window size were cut into several windows.

Following experimentation with different window sizes, it was concluded that a higher window size can capture a larger part of a user's interaction history and therefore generally has better performance. However, high window sizes proved to be slow to train owing to larger matrices which were slow to multiply.

After some experimentation with windows of varying lengths, a window size close to the 90th percentile of user history length proved to be good enough with regards to training time and model performance for most datasets.

Initially, the windows were shifted one step at a time to maximize the number of time sequences that could be used for training. This meant that users with long exercise histories would end up creating many overlapping windows, providing more data for training. However, this proved infeasible for larger datasets like EdNet due to limited hard disk space and long training times.

In addition, this would cause users with a very long exercise history to have disproportionate representation in the data and introduce unwanted bias to the model. Therefore a *window stride* was introduced which decided how many steps the window would be shifted each time when windowing. After some experimentation the window stride was set to half the window size to create sequences with 50% overlap as a compromise between size and performance.

4) Training and validation: We split each dataset into a training and validation set with a 95:5 ratio.

5) Types of features: We have decided to categorize the input features into two categories, depending on what type of information they hold, in order to more easily explain their usage in the models.

Query features corresponds to information about the current exercise $\mathbf{e_t}$ and consists of the unique identifier and the category of the exercise currently being answered, in addition to the mean response time and correctness ratio for questions with the same identifier.

Memory features consists of information about the past response $\mathbf{r_{t-1}}$, and consists of the correctness, response time and the timestamp difference of the response.

In addition, attention-based models benefit from *positional embeddings*, which include information on the positions of the data points in a sequence. This is useful because attention mechanism, unlike a recurrent neural network, does not operate sequentially on time-series data and therefore does not keep track of the order of data points in a sequence. Therefore, positional embeddings are added to both the query features and the memory features in attention-based models.

6) Feature engineering: To gain as much information as possible from the relevant data in each dataset, we engineered a series of features for each user interaction in users' exercise histories.

a) Time difference: The first engineered feature is the time that has passed between each response and the response before it. This feature can effectively track how long it has been since the user last responded to a question.

The motivation for including this feature is that students tend to forget information over time. By looking at how long it has been since a student last solved exercises, the model could learn whether a student had forgotten some information and would likely perform worse.

The feature was created by calculating the difference between the submission timestamps of each question and the question before it. In addition, the current question's response time was subtracted from this value to get the actual time difference between submitting the answer to a question and starting the next question. This is similar to the *lag time* feature used by [14].

b) Mean response time per exercise: While response time is individual and dependent on the fluency of the student, it is also highly dependent on the characteristics of the exercise itself. In other words, some exercises take a longer or shorter time to solve, regardless of the user's fluency.

This can depend both on the question's difficulty and the time it takes to formulate the question to the student. An example of the latter would be a question, which has a long text description that takes a long time to read. By supplying the model with the mean response time for each unique exercise, the model should be able to take this into account.

c) Standard scored (z-scored) response time: Response time has often been used as a feature in previous knowledge tracing studies. Past studies simply used the response time in seconds as a new feature.

We hypothesized that if the model is given information about how the user performed *relative to other users*, important information about the fluency of the user can be inferred.

One way to implement such a feature would be to use standard scoring over all response times in the dataset. However, as mentioned before, different exercises take different amount of time to complete on average. Since the mean response time is different for each exercise, the response time was normalized based on responses to the same exercise ID. We started by first calculating the mean response time μ and standard deviation σ for each exercise. Afterwards, the standard score z for each response was calculated using

$$z = \frac{x - \mu}{\sigma},$$

where x is the response time in seconds.

This standard score is equivalent to how many standard deviations a single observed response time is from the mean, when considering every other response time to the same question. Therefore, a positive value would mean a slower than average response to the question and a negative value would mean a faster than average response time.

d) Mean correctness per exercise: The mean correctness of an exercise is calculated by dividing the amount of correct responses with the total responses for that exercise. Expressed as a continuous value between 0 and 1, the mean correctness can serve as a measure of difficulty for each exercise.

7) Continuous embeddings: There are features in this study that are not categorical, namely response time (and its standard-scored variant), timestamp difference, mean response time and mean correctness. Normally, these features could be concatenated to the rest of the embeddings along the feature dimension. However, when using additive embeddings similar to those described in [11] and [13], this is not possible, as the features that are added need to have the same dimensions.

These features could be cast into integers and turned into embedding vectors with correct dimensions using random embeddings. However, the random nature of the embeddings would prevent the model from utilizing the continuous nature of the values. For example, the random categorical embeddings for an response time value of 4 seconds would on average have the same similarity to the embeddings for 5 seconds and the embeddings for 50 seconds. In reality, 4 seconds and 5 seconds are much closer to each other than to 50 seconds.

a) Continuous embeddings: In order to create embeddings that preserve this fundamental continuity in continuous values, a special type of embeddings known as *continuous embeddings* similar to those used by [14] were employed. These embeddings are trainable vectors with a length equal to that of the categorical embeddings. These vectors are multiplied by their respective continuous-valued features to create embeddings with the same length as the categorical embeddings. Thanks to this property, the continuous nature of the features are preserved since they are multiples of the same embeddings vector. Shin et. al have shown that for the SAINT model and for the response time feature, which takes on continuous values, continuous embeddings result in better performance compared to categorical embeddings [14].

B. Models

A series of different knowledge tracing models were implemented and benchmarked, namely Deep Knowledge Tracing (DKT) [2], Self-Attentive Knowledge Tracing (SAKT) [11], Separated Self-Attentive Neural Knowledge Tracing (SAINT) [13] and Last Query Transformer RNN (LQTR) [16]. The first model is purely RNN-based, the next two are attention-based models, and the last model is a hybrid model that utilizes both attention mechanism and an RNN layer. We implemented all models from code provided by the authors or, in case such code is unavailable, by attempting to implement the model architectures as described by the authors.

We ran each model according to the hyperparameters given by their authors, when available. Otherwise, we used manual hyperparameter tuning. Since most of the models had not been run on most of the datasets in previous studies, it was necessary to run many tests to manually optimize the hyperparameters for each model-dataset pair.

For every model, the input features were transformed into a series of embeddings and concatenated. These embeddings were then fed into a feed forward network, or FFN (also known as a *multilayer perceptron* or MLP) [18] to reduce the total dimension of the input to the dimension of a single embedding, before being input into the models.

The original models which predicted correctness featured a sigmoid activation function in the final layer, and were trained using a binary cross entropy loss function.

In order to adapt these models for response time prediction, we have removed the sigmoid activation function from the final layer and used mean squared error loss for training.

1) DKT: Deep Knowledge Tracing is an LSTM-based model first proposed in [2]. It is the simplest model used in the study and consists of an LSTM layer followed by an FFN.

While the original DKT model uses an encoded set of tuples representing each posssible exercise tag and correctness value (correct or incorrect) as input features, the version of DKT used in this paper is modified to also incorporate temporal features (response time and timestamps), as well as additional features (mean correctness and mean response time). All of the the query and memory embeddings are concatenated and passed through an FFN before being passed to the LSTM layer.

The output of the LSTM layer is then fed into an FFN which makes the final prediction.

2) SAKT: Self-Attentive Knowledge Tracing is an attention-based knowledge tracing model first proposed in [11]. It uses one or more multi-headed attention layers [10], and an output FFN layer. Since attention models are not constrained by sequentiality during training, the SAKT model is very fast to train. The implemented variant of SAKT also uses the additional temporal features described earlier in the *feature engineering* section. The query for the multi-head attention model consists of the query feature embeddings passed through an FFN and the key and value consist of the memory features passed through a different FFN for key and value.

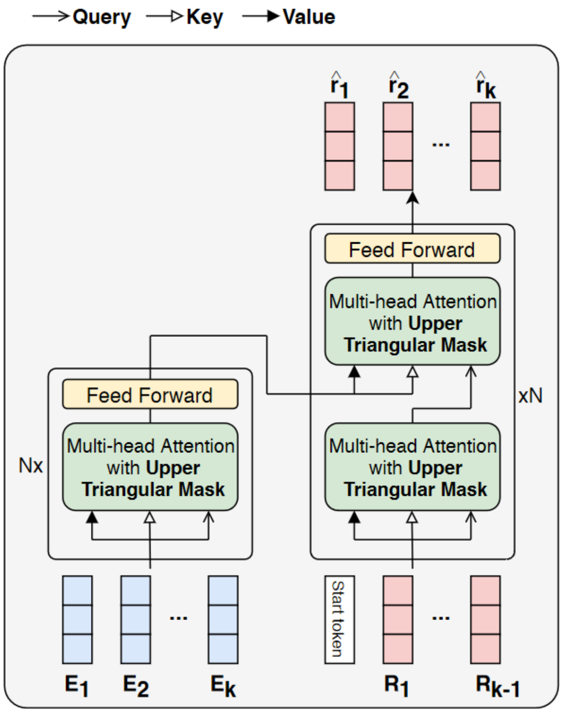

Query
Key
Value
$\hat{r}_1$
$\hat{r}_2$
...
$\hat{r}_k$
Feed Forward
Multi-head Attention with Upper Triangular Mask
xN
Feed Forward
Multi-head Attention with Upper Triangular Mask
Nx
Multi-head Attention with Upper Triangular Mask
E_1
E_2
...
E_k
Start token
R_1
...
R_{k-1}

Fig. 2: SAINT architecture from [13]

3) SAINT: Separated Self-Attentive Neural Knowledge Tracing is a Transformer-based model described in [13]. It uses a transformer model with a variable number of *encoder* and *decoder* layers [10], and an output FFN layer, as seen in Figure 2. The encoder and decoder layers have different inputs: The encoder uses the memory embeddings as query, key and value, and the decoder uses the query embeddings as query and the encoder output as the key and value. Our variant of SAINT is more similar to the proposed SAINT+ model described by [14] owing to its use of temporal features.

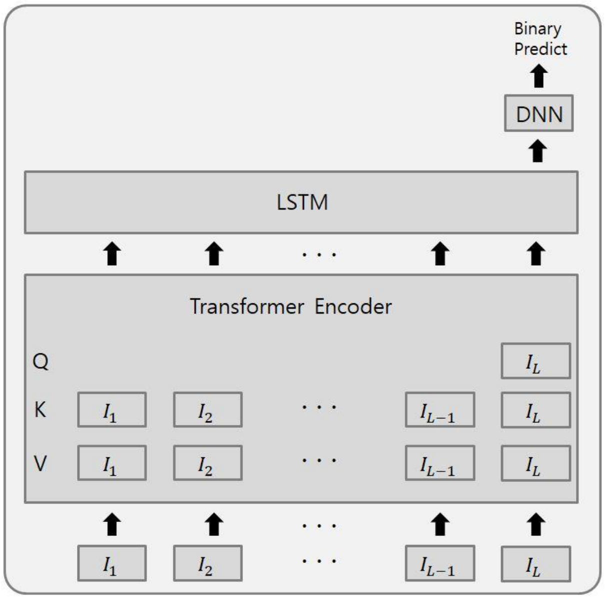


Fig. 3: LQTR architecture from [16].

4) LQTR: Last Query Transformer RNN is a hybrid model consisting of a SAKT-like multi-headed attention layer connected to an LSTM layer and finally an output FFN layer. It was proposed by [16] and won the *2020 Riiid AIEd Challenge* on Kaggle. The architecture, as seen in Figure 3, takes a sequence of L interactions, I_1, I_2, ..., I_L, which are then input into a single Transformer encoder. The output of the encoder passes through an LSTM into an FFN (Feed Forward Network) which outputs the prediction.

The LQTR model is noteworthy for a few innovations in knowledge tracing architecture design, described in the sections below.

a) The Q-trick: The main novelty of the architecture pertains to how attention is calculated. Traditionally, attention is calculated by matrix multiplication [10],

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

where the query matrix, Q, is multiplied with the transposed key matrix, K^T. The operation has a complexity of $O(L^2)$ where L is the sequence length. The author realized that using only the last query vector, instead of a matrix containing each query vector in the sequence, yielded minor performance losses while reducing the complexity of the matrix operation to $O(L)$. This optimization, called the Q-trick, allowed the author to substantially increase the sequence length which yielded a net performance increase, even counting the loss resulting from the optimized matrix multiplication [16].

b) Capturing sequence-related patterns with an LSTM: Whereas the output of transformer-based models are usually routed through an FFN before being output as a prediction, LQTR routes the output of the transformer through an LSTM before routing it through a FFN. Thus, the transformer encoder is responsible for capturing relationships between questions, and the LSTM is responsible for finding sequence-related patterns in the data. Intuitively this can be understood as the encoder supplying the LSTM network with its understanding which is based on the relationships between every question and the LSTM network will use this information to find meaningful sequential patterns, which the encoder cannot do by itself. In this sense, the LSTM layer effectively has a role similar to the decoder in a Transformer neural network.

V. Experimental settings

A. Code and implementation

When available, we used the author's code for the models. However, some of the papers lacked publicly available code. In some other papers, the code was written in old versions of frameworks which could not be run on current Python environments or did not work with current *CUDA* versions. In these cases, we re-implemented the models in PyTorch 1.7.1, following the description of the model given in the respective model's paper as closely as possible.

All code is available at a GitHub repository[1].

B. Training

For training, we used several datasets in addition to the Akribian dataset, as described in chapter 4. We ran each model on every dataset and logged the results for each run. All models were trained on one *Nvidia RTX 2070* GPU. The optimizer used for training was Adam with a step size of 10^{-3} and β values of $(0.9, 0.999)$.

In some cases, we weren't able to match the authors' reported results due to a lack of transparency regarding the code, the hyperparameters and the datasets for the models. In addition, some models had high variance in their final accuracy for each dataset, which implies that better results are likely possible given a large enough number of training attempts.

C. Hyperparameters

For models whose hyperparameters are described in their paper, we tested the mentioned hyperparameters. We also used manual hyperparameter tuning to find hyperparameters that offered accuracy close to the author's reported accuracy, while also taking care to not increase training time too much.

We eventually settled on using a latent dimension of 128, 2 attention heads, 2 layers of encoders, 2 layers of decoders and 2 LSTM layers for all tests. We used a default batch size of 64, but we adjusted this value depending on available VRAM, specifically decreasing it when running on datasets with large window size.

[1] https://github.com/sepehrnoor/response-time-prediction

D. Loss functions

When predicting correctness, binary cross entropy defined as

$$L = -\frac{1}{N}\sum_{i=1}^{N}\ell(y_i, \hat{y}_i)$$

is used as a loss function, where the loss L is dependent on the ground truth correctness y_i and the predicted correctness $\hat{y}_i$ and

$$\ell(y, \hat{y}) = y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i),$$

was used as a loss function.

When predicting response time, mean square error (MSE) defined as

$$L = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2,$$

was instead used as a loss function.

E. Dropout

To prevent overfitting to the training data, we used dropout. After testing different values, a dropout rate of 0.2 showed the largest increase in validation AUC and was therefore chosen as the default value.

F. Evaluation

We split each dataset into a training and a validation dataset with a 95:5 ratio. Instead of shuffling the datasets, each dataset was split into 20 sequential segments, and each segment was split and concatenated into the training and validation datasets. This showed to be faster and give better results for most of our datasets.

After training, each model was evaluated on the validation data sequences using the corresponding evaluation criterion for each task.

1) Evaluation Criterion: In order to measure and compare models within the tasks of correctness and response time prediction suitable evaluation criterion are needed. Whereas correctness prediction is a binary classification problem with two classes (correct and incorrect), response time prediction is a regression problem with a continuous range of possible values. Because of this inherent difference between the tasks, two different approaches have been used.

a) Evaluating correctness prediction: Since correctness prediction is a binary classification problem, when comparing two models the better model is able to better separate correct and incorrect answers. Area under curve (AUC) effectively measures this separation by using a Receiver Operating Characteristic (ROC) curve which plots the ratio between the True Positive Rate (TPR) and False Positive Rate (FPR) for each possible threshold value.

Whereas a model's predictions range in a continuous range from 0-1, a perfect model would, for some arbitrary threshold value such as 0.5, be able to correctly output a value below 0.5 for incorrect answers and output a value over 0.5 for correct answers. This perfect separation would yield an AUC score of 1, whereas a model that is not able to achieve any separation at all would yield a score of 0.5.

b) Evaluating response time prediction: For regression problems mean absolute error (MAE) and coefficient of determination (R^2) are among the most common evaluation metrics.

MAE [19] is an intuitive metric since it can be measured in actual time units and is thus suitable for comparing the performance of different models on the same dataset. Since MAE is dependent on the response time distribution of each dataset, it is not as suitable for evaluating models across different datasets.

The R^2 score, which measures the squared correlation of the predicted value and the ground truth, is on the other hand more suitable for comparing different datasets.

VI. Results and discussion

A. Correctness prediction

The best results for each model and dataset can be seen in Table II. In addition to the listed models, a simple statistical baseline was created for comparing model performance.

The baseline consists of a *dictionary* that returns the mean correctness (the ratio of correct answers per unique exercise) of each unique exercise in the training dataset, without using any machine learning methods.

	ASSISTments	**Junyi Academy**	**EdNet**	**Akribian**
Baseline	0.720	0.688	0.717	0.830
DKT	**0.980**	0.793	0.770	**0.965**
SAKT	0.758	0.757	0.753	0.917
SAINT	0.741	0.767	0.764	0.919
LQTR	0.948	**0.794**	**0.792**	0.940

TABLE II: Table of models' AUC score for correctness prediction across datasets.

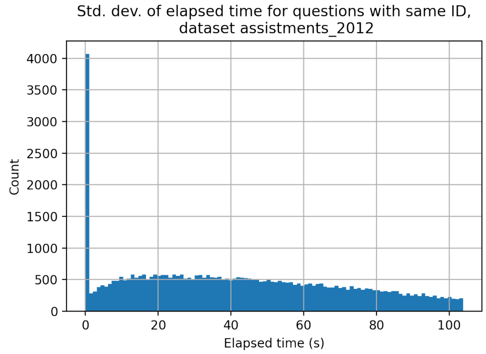


Fig. 4: Histogram of response time standard deviation for ASSISTments 2012 dataset.

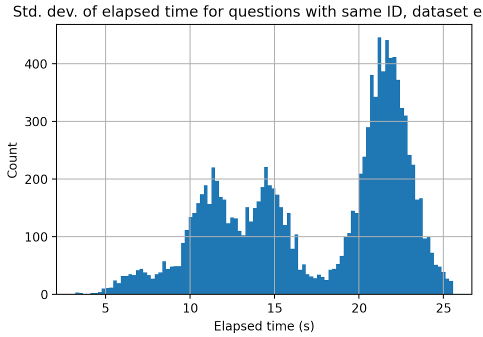


Fig. 5: Histogram of response time standard deviation for EdNet dataset.

While DKT and LQTR clearly outperform SAKT and SAINT on the ASSISTments and Akribian datasets, model performance is more equal on the other datasets.

Figure 4 shows a histogram of the response time standard deviations calculated for each exercise category for the ASSISTments 2012 dataset. Figure 5 shows the same histogram for the EdNet dataset. It can be seen that a significantly higher degree of exercises with a standard deviation of zero can be observed in the ASSISTments dataset compared to the EdNet dataset. That is, for a large set of exercises in ASSISTments 2012, every student has answered uniformly according to the mean for that exercise. The reason for this is that the ASSISTments dataset has comparatively fewer entries per unique exercise than the Junyi and Ednet datasets. This issue is also apparent in the Akribian dataset; ASSISTments and Akribian have on average 52 and 91 entries per unique exercise while Junyi and Ednet have on average 12221 and 7485 entries per exercise, respectively.

With so few entries per exercise, it might be the case that SAKT and SAINT, both being models purely based on attention, do not have enough data to meaningfully model relationships between exercises. DKT and LQTR, on the other hand, may be able to leverage the nature of their LSTM components to forego relationships between exercises and instead focus on input features and the short-term performance of each student.

Because DKT and SAKT used datasets without time features we were not able to compare our performance directly to the original papers, however our performance was in-line with their self-reported relative performance. We found a 0.017 and 0.028 loss in AUC compared to the original papers for SAINT and LQTR, respectively. We deem this to be an acceptable loss considering time and resource limitations.

B. Response time prediction

The best results for each model and dataset can be seen in Tables III and IV. A baseline was created by calculating the mean response time for each exercise in the training set. For each exercise in the test set, the calculated mean response time is used as the baseline's prediction.

	ASSISTments	Junyi Academy	Ednet	Akribian
Baseline	10.65	13.91	6.40	1.30
DKT	10.67	13.95	**3.465**	**0.572**
SAKT	10.56	12.96	4.56	0.60
SAINT	12.21	**12.951**	5.35	0.74
LQTR	**9.350**	13.68	4.03	0.58

TABLE III: Table of models' MAE score (in seconds) for response time prediction across datasets

	ASSISTments	Junyi Academy	Ednet	Akribian
Baseline	0.264	0.176	0.128	0.574
DKT	0.494	0.156	**0.639**	**0.965**
SAKT	0.533	0.227	0.474	0.888
SAINT	0.459	**0.235**	0.361	0.856
LQTR	**0.580**	0.187	0.610	0.920

TABLE IV: Table of models' R^2 score for response time prediction across datasets

It can be seen that the results vary between datasets, with two of the datasets achieving best results using the simpler DKT model. We believe this could mean that response time is more dependent on short-term patterns in user behaviour, since RNN-based models are designed to find short-term patterns in data, as opposed to attention which is better at modeling long-term relations.

Furthermore, the machine learning models are able to achieve a significant performance increase compared to baseline for the EdNet and Akribian datasets. This means that the models are learning meaningful information from the data. While the other datasets have not outperformed the baseline models to the same degree, the results show that there actually is meaningful patterns in the data that can be used for response time predictions.

1) Feature study: We ran two separate features studies using different datasets on the LQTR model to investigate how different features affect performance on different datasets.

The effects of adding individual features to the LQTR model when predicting the Junyi Academy and ASSISTments datasets can be seen in Tables V and VI, respectively. Since these feature studies were performed at different times during the paper, there are differences in the premise of the studies. The baseline for Junyi Academy consists of ID embeddings, exercise category embeddings and random positional embeddings. The baseline for ASSISTments includes past correctness embeddings in addition to the aforementioned embeddings.

TABLE V: Effect of features on performance of LQTR response time prediction model on the Junyi Academy dataset.

	Baseline	**Baseline + C**	**Baseline + R**
MAE (s)	14.27	14.18	13.27
MAE difference	–	-0.09	-1.00
	Baseline + Z	**Baseline + T**	**Baseline + S**
MAE (s)	13.26	14.20	14.04
MAE difference	-1.01	-0.07	-0.23
	Baseline + MR	**Baseline + MC**	
MAE (s)	14.41	14.51	
MAE difference	+0.14	+0.24	

TABLE VI: Effect of features on performance of LQTR response time prediction model on the ASSISTments 2012 dataset.

	Baseline	**Baseline + Q**	**Baseline + R**
MAE (s)	13.92	14.24	14.64
MAE difference	–	+0.32	+0.72
	Baseline + Z	**Baseline + T**	**Baseline + S**
MAE (s)	12.47	12.82	12.75
MAE difference	-1.45	-1.10	-1.17
	Baseline + MR	**Baseline + MC**	
MAE (s)	11.06	12.06	
MAE difference	-2.86	-1.86	

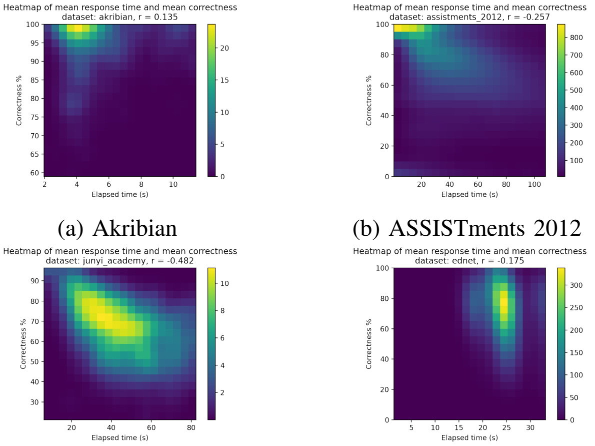
^{C} Past correctness embeddings.
^{Q} Q-trick.
^{R} Response time.
^{Z} Z-scored response time.
^{T} Timestamp difference.
^{MR} Mean response time.
^{MC} Mean correctness.
^{S} Sine positional embeddings.

Lending to the differences between the datasets, as well as the fact that the baselines are different, some features may improve performance in one of the datasets, but not the other. For example, mean response time and mean correctness give a major performance increase for the ASSISTments dataset, while actually decreasing performance for the Junyi Academy dataset. This may be caused by the higher response variance of the Junyi Academy dataset.

A feature that has yielded a major performance increase for both datasets is z-scored response time. Interestingly, it works better than response time itself.

Time-stamp difference, which is often used to model forgetfulness, also gives a minor performance increase. It is however likely that the model is actually learning how to deduce response time from it when used as as single feature. Its suitability as a feature should thus be measured in its capacity to increase performance when added to a model already using z-scored and normal response time.

Sine positional embeddings also give a minor performance increase since the embeddings are designed to be preserve some notion of continuity between subsequent positions. This makes it easier for the attention-based LQTR model to understand the order of the input data.



(a) Akribian

(b) ASSISTments 2012

(c) Junyi Academy

(d) EdNet

Fig. 6: 2D histogram of mean response time per question and correctness ratio. Mean response time values above the 90th percentile have been removed for clarity.

The performance gain from including mean correctness is not as intuitive to explain for the response time prediction model. We speculate that it might be related to the fact that mean correctness and mean response time are not independent of each other. In other words, we believe it is likely that easy questions would take less time to answer and more difficult questions would take longer to answer. Statistically, this would imply an inverse correlation between response time and correctness. To confirm or deny this hypothesis, we conducted a statistical analysis, the results of which are visible in Figure 6. It can be seen that exercises in most datasets, especially Junyi Academy, have a slight negative correlation between mean response time and mean correctness. The exception is Akribian, which instead has a very slight *positive* correlation.

This difference could be thanks to the difference in the nature of the data in the Akribian dataset. The data in the Akribian dataset comes from an educational game for young children, whereas the data for the other datasets comes from e-learning platforms for older demographics.

On the Junyi Academy dataset, these features decreased model performance. This could be because these features may not provide useful information, thanks to the high variance in the response times in the Junyi Academy dataset. This is in contrast to the low response time variance found in the ASSISTments 2012 dataset.

Finally, we also tried different combinations of features, but no combination yielded better performance than the best individual features.

C. Comparison between response time and correctness prediction

When comparing Tables II and III, it can be seen that different models have varying performance on different dataset and there is no model that performs best on every dataset.

When predicting correctness, LQTR performs better on larger datasets while DKT performs better on the smaller ones. However, when predicting response time, DKT – which is an LSTM-based model – performs better on the large EdNet dataset, as well as the small Akribian dataset. LQTR performs better on the ASSISTments 2012 dataset, and SAINT performs better on Junyi Academy dataset.

This disparity could be caused by the fact that the task of response time prediction is more dependent on short-term connections than the task of correctness prediction. When predicting correctness, it is after all important to know whether a student has answered correctly on past questions with related concepts. On the other hand, response time may be more closely tied to the current pace of the student than their knowledge of related concepts.

If this is the case, then the comparatively better performance of DKT is not surprising, considering that the advantage of attention in comparison to LSTM is the ability to model long-term relationships.

VII. Conclusion and future work

This paper aimed to adapt existing models within the field of knowledge tracing to the task of predicting the exercise response time of students using online learning platforms. By benchmarking several models and datasets and comparing to simple statistical baselines, we achieved promising initial results.

By studying the effect of different features on the performance of the LQTR model in the task of knowledge tracing, we showed that certain features yielded a significant performance increase. In particular, two features that we engineered, namely mean correctness and z-scored response time, yielded a major performance increase when predicting correctness.

Furthermore, a comprehensive comparison of the largest datasets and most common models yielded insights into why certain models may work better than others for a particular dataset. In particular, we found that the differences in performance were higher across datasets than models. Bigger datasets that have a lot of answers for each unique exercise were harder to predict than smaller datasets. The characteristics of the dataset, in particular the variance in the predicted variables, must be taken into account. When dealing with datasets with low response time variance in correctness and response times, for example, mean correctness and mean response time features provide large performance increases.

Furthermore, LSTM has performed better on datasets with low variance whereas LQTR, an attention-based model, performed better on the datasets with high variance.

Due to the decision making process of machine learning models being inherently difficult to interpret, we don't actually know whether or not the model's are learning meaningful representations of student knowledge. Because of this, we believe that further research should be conducted in order to interpret the decision making process of current algorithms.

In the end, knowledge tracing predictions will have to be integrated with educational models in order to actually be applied in the real world. When doing so, we believe that the addition of response time prediction will allow for more nuanced models, especially for quantifying fluency which is highly correlated with speed. We hope that our models can find applications in learning path recommendations in e-learning platforms.

References

[1] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User Modelling and User-Adapted Interaction*, vol. 4, no. 4, p. 253–278, 1995.

[2] C. Piech, J. Spencer, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein, "Deep knowledge tracing," *CoRR*, vol. abs/1506.05908, 2015. [Online]. Available: http://arxiv.org/abs/1506.05908

[3] L. Fuchs, D. Fuchs, D. Compton, S. Powell, P. Seethaler, A. Capizzi, C. Schatschneider, and J. Fletcher, "The cognitive correlates of third-grade skill in arithmetic, algorithmic computation, and arithmetic word problems." *Journal of Educational Psychology*, vol. 98, pp. 29–43, 02 2006.

[4] S. L. Decker and A. M. Roberts, "Specific cognitive predictors of early math problem solving," *Psychology in the Schools*, vol. 52, no. 5, pp. 477–488, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/pits.21837

[5] D. E. Rumelhart and J. L. McClelland, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MIT Press, Cambridge, MA, 1987, pp. 6–7.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[7] J. Zhang, X. Shi, I. King, and D. Yeung, "Dynamic key-value memory network for knowledge tracing," *CoRR*, vol. abs/1611.08108, 2016. [Online]. Available: http://arxiv.org/abs/1611.08108

[8] Q. Liu, Z. Huang, Y. Yin, E. Chen, H. Xiong, Y. Su, and G. Hu, "Ekt: Exercise-aware knowledge tracing for student performance prediction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 1, pp. 100–115, 2021.

[9] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, 2015.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[11] S. Pandey and G. Karypis, "A self-attentive model for knowledge tracing," in *Proceedings of The 12th International Conference on Educational Data Mining (EDM 2019)*, 2019.

[12] S. Pandey and J. Srivastava, "Rkt: Relation-aware self-attention for knowledge tracing," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. ACM, Oct 2020. [Online]. Available: http://dx.doi.org/10.1145/3340531.3411994

[13] Y. Choi, Y. Lee, J. Cho, J. Baek, B. Kim, Y. Cha, D. Shin, C. Bae, and J. Heo, "Towards an appropriate query, key, and value computation for knowledge tracing," in *Proceedings of the Seventh ACM Conference on Learning @ Scale*, ser. L@S '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 341–344. [Online]. Available: https://doi.org/10.1145/3386527.3405945

[14] D. Shin, Y. Shim, H. Yu, S. Lee, B. Kim, and Y. Choi, "Saint+: Integrating temporal features for ednet correctness prediction," in *Proceedings of the 11th International Learning Analytics and Knowledge Conference (LAK21)*. New York, NY, USA: Association for Computing Machinery, 2021, p. 490–496.

[15] Y. Choi, Y. Lee, D. Shin, J. Cho, S. Park, S. Lee, J. Baek, C. Bae, B. Kim, and J. Heo, "Ednet: A large-scale hierarchical dataset in education," in *Artificial Intelligence in Education*, I. I. Bittencourt, M. Cukurova, K. Muldner, R. Luckin, and E. Millán, Eds. Cham: Springer International Publishing, 2020, pp. 69–73.

[16] S. Jeon, "Last query transformer RNN for knowledge tracing," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI-21)*, 2021.

[17] M. Feng, N. Heffernan, and K. Koedinger, “Addressing the assessment challenge with an online system that tutors as it assesses,” *User Modeling and User-Adapted Interaction*, vol. 19, no. 3, pp. 243–266, Aug 2009. [Online]. Available: https://doi.org/10.1007/s11257-009-9063-7

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, p. 318–362.

[19] “Mean absolute error,” in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 652–652. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_525