OXFORD JOURNAL
OF STUDENT
SCHOLARSHIP

Improving the Efficiency of Knowledge Tracing Algorithms: LinSAKT

Izhan Ali
izhan.ali08@gmail.com

ABSTRACT

Knowledge sharing through teaching is an essential aspect of human intelligence. With the advent of online teaching platforms, teachers have the ability to track student progress and personalize teaching to tailor their learning experience. Predicting a student's performance by tracking their learning progress and interactions with the study material is called the Knowledge tracing (KT) problem. The KT problem in general has found usage in intelligent tutoring systems (ITS), curriculum learning and other computer-aided learning tools. Due to the availability of massive datasets, KT problem is effectively solved using various deep learning algorithms. Each of these algorithms have exploited the deep KT (DKT) problem through memory-augmented neural networks, attention mechanisms, graph-based networks and feature engineering techniques. These techniques drastically improve the accuracy of these algorithms making them more suited for various teaching related tasks. In this research, we focus on one of the well-known DKT algorithms called SAKT (attention-based framework) and study its computational efficiency and accuracy. Attention based frameworks (SAKT) have higher accuracy, incur higher memory footprint and greater runtime. To take advantage of their accuracy but lower the runtime and memory usage, we propose a linear attention model for the KT task called LinSAKT. This model preserves the contextual information captured by the attention mechanism through Linear attention and significantly reduces the runtime and memory footprint. This model is a relevant addition to solving the KT problem while enhancing efficiency. Three datasets were used to study efficiency compared to the baseline SAKT. LinSAKT reduced training time and memory usage. Averaged across three ASSIST datasets and all tested hyperparameter configurations (36 paired comparisons total), LinSAKT reduces training time per epoch by 14.1% and peak memory per epoch by 30.6% (unweighted mean of per-configuration percent reductions). LinSAKT enables KT in low-resource environments supporting equitable access to AI-driven personalized learning tools in underserved regions.

INTRODUCTION

Online learning systems are a common aid to teaching and learning especially in the post-COVID era. These systems provide a flexible learning environment for students and an ability to trace student learning patterns for educators. These systems allow personalization through vast amounts of data collected through a student's learning process and can therefore be tailored to the needs of students. Various techniques are used to model these systems adopted from multiple fields like Machine Learning (ML), Artificial Intelligence (AI), Cognitive Science etc. (Abdelrahman et al., 2023). In the development of these systems knowledge tracing (KT) is an important task and is defined as the task of tracing a student's knowledge level determined through their mastery of knowledge concepts (KC) based on historical learning activities. Under the ML and AI paradigm, the problem is formulated as a supervised learning problem (Wang et al., 2016) where the input (X) is the sequence of past exercise interactions and student response to each exercise. The goal of KT is to predict whether the student will answer a given exercise correctly.

In recent years, deep learning (DL) models like Recurrent Neural Networks (RNN) and their variants are used to model this problem (Piech et al., 2015, Li et al., 2021). Most of the RNN based models suffer from generalization on sparse data which is a common problem with most publicly available datasets. To address this problem, transformer-based models (Vaswani et al., 2017) were used to solve the KT problem. The self-attentive knowledge tracing (SAKT) model is one such model that outperformed state-of-the-art models for KT (Pandey & Karypis, 2019). It was found to be faster than RNN based models. One key component of the SAKT model is the self-attention mechanism which is the core of any transformer-based model. This is a bottleneck for efficiency as specified in various other studies (Wang et al., 2020, Liu et al., 2023). SAKT uses attention mechanism to retain contextual information of the sequence of exercises attempted by students and the associated KCs. This requires retaining information about all past exercises while predicting student performance. Each exercise is represented using a numerical value called token. Each token representation is updated by using all other tokens in the previous layer in SAKT model. The attention score computation scales as $O(N^2 d)$ time and $O(N^2)$ memory (attention matrix), where sequence length (number of past exercises considered - denoted by N) and embedding dimensions (the dimensionality of token representations used - denoted by d).

In this work a novel approach for modeling the self-attention bottleneck in SAKT is introduced. This approach is inspired by the linear attention mechanism described in (Liu et al., 2023) with some computational adjustments. We propose LinSAKT, a SAKT-style knowledge tracing model that replaces scaled dot-product attention with causal linear attention. While recent work has explored efficiency-oriented transformer variants for knowledge tracing (e.g., Lite-Transformer-style approaches – (Liang, Z. et al, 2024), (Yao, S. et al, 2025)), LinSAKT provides a simple architectural substitution and an empirical study of runtime and GPU memory scaling with sequence length across multiple ASSIST datasets.

Contributions of this research are:

1. In this research, an efficient self-attention mechanism for the KT problem is developed. This new model is called LinSAKT.
2. LinSAKT reduces the complexity of the model making it linear in sequence length by replacing the dot-product attention in the original model to a normalized linear attention model.
3. An in-depth empirical analysis was carried out on three well-known datasets to establish the effectiveness of LinSAKT. The experiments indicated superior performance in terms of runtime and memory footprint compared to the dot-product based SAKT model.

4. LinSAKT exhibited comparable accuracy across various evaluation metrics (Recall, Precision and F1-Score).
5. This model provides a blueprint for the creation of AI-driven personalized tools that are easy to scale and deploy, thereby supporting low-resource environments.

BACKGROUND

Dot Product Attention

In this research, a KT model was developed which uses the idea of a Transformer (Vaswani et al., 2017). The most important part of a Transformer is the attention mechanism. This mechanism works on the idea of extracting relevant contextual information in a sequence by placing more emphasis on the token that carries more information. This is achieved through scaled dot-product attention. If n is the length of a sequence and d is the size of the embedding (vector used to describe a token), then attention mechanism is given by the following:

$$A = softmax(\frac{QK^T}{\sqrt{d}})V$$

Here $Q = XW^Q$, $K = XW^K$, $V = XW^V$, where $X \in R^{n\times d}$ is the matrix of input sequence. All W's are weight matrices where learnable parameters called projections will be saved during the training process. $Q, K, V \in R^{N\times d}$ are called query, key and value matrices (Vaswani et al., 2017). The SoftMax function is applied row-wise and the division by square root of d is for scaling. A is called the attention mechanism.
The attention mechanism has been useful in capturing long-term dependencies in sequences. It addresses the vanishing gradient problem that is an issue in RNNs. However, its calculation involves $n \times n$ computation making it quadratic in sequence length. Usually, the embedding dimension is assumed to be much smaller than n, therefore not contributing much to the complexity. However, if d is large then the attention calculation involves a quadratic complexity of n and d.

Knowledge Tracing Task

Knowledge Tracing (KT) is the process of modeling student's knowledge state by analyzing a sequence of their interactions with educational material. The task models a student's progression over time accounting for both learning and forgetting over time. The temporal order of interactions is critical in modeling this task. There are three major models used in modeling the KT problem: 1) Bayesian models 2) Logistic Models 3) Deep learning models (DKT) (Shen et al., 2024). This research focuses on DKT due to their ability to capture the student cognitive process better than the other two (3). Deep learning models in general can capture non-linearity better and are well suited for modeling complex learning processes (Mandalapu et al., 2021). Initially DKT involved usage of RNNs to model KT. However, with the advent of Transformers in the Natural Language Processing (NLP) space it was found that sequential dependencies can be better captured by these models (Devlin et al., 2019, Brown et al., 2020). Because NLP task is similar to KT task due to its temporal nature, these models found usage in this field. SAKT was one of the first models that involved self-attentive model for KT (Pandey & Karypis, 2019) and has been found to be effective for a range of KT tasks.

Linear Attention Models

Recent research in Transformer models emphasizes 'efficiency', which is defined in a number of different ways. Anything impacting the cost of the model contributes to efficiency for example Floating-point Operations, inference time, runtime, model parameters, energy consumption and carbon emissions (Ansar et al., 2024). Several Linear Transformer models have been proposed to address one or more of the cost issues with the goal of improving efficiency (Wang et al., 2020, Zheng et al., 2022, Qin et al., 2022). These models have been adopted in various NLP based tasks and are shown to reduce time and/or memory footprint.

There has been limited advancement in Transformer based approaches that reduce cost (improve efficiency) in the DKT field. This is one of the early studies that aims to address this problem. Efficient KT models are useful because these can be used to create lightweight applications leading to better reach and contributing to the growing field of Education Technology. Reducing memory and energy usage may be associated with carbon footprint and environmental impact (Bannour et al., 2021). LinSAKT, the model introduced in this research is a step towards addressing these issues. Reducing runtime and memory footprint can be a practical step toward more resource-efficient machine learning deployments. In this study, we quantify efficiency using training time per epoch and peak GPU memory usage. We do not directly measure energy consumption or carbon emissions. Therefore, environmental impact is a potential downstream implication rather than a demonstrated outcome of our experiments.

METHODOLOGY

Problem Statement

For any KT problem student interactions are captured as a sequence of problems that a student attempts where the input at each time-step is the question/exercise and output is the correctness of response. These sequences are transformed into a sequential modeling problem. Considering a finite number of student interactions, an interaction at time step t is denoted by $x_t = (q_t, r_t)$, where q_t is the question attempted at time step t and $r_t \in \{0, 1\}$ is the correctness of student response. The model receives the interaction value as a single number $y_t = q_t + r_t \times n\_skill$, where n_skill is the total number of skills in a given set of problems (dataset).

The LinSAKT Model

The LinSAKT model is a computationally-efficient variant of the Self-attentive Knowledge Tracing (SAKT) model introduced in (Pandey, S. et al., 2019). LinSAKT follows the original SAKT data preprocessing and training pipeline; we only replace scaled dot-product attention with causal linear attention. LinSAKT replaces the standard multi-head self-attention with a causal linear attention mechanism to achieve linear time and space complexity. The model maintains predictive performance in knowledge tracing task involving sequential interactions. LinSAKT solves the KT problem by predicting the probability of correctly answering future questions based on historical question-answer response. The model utilizes input sequences of question-response encodings (numeric representations in a vector space)

$X = (x_1, x_2, x_3 \ldots x_T)$ where T is called the maximum sequence length (representing total number of interactions possible).

The model architecture consists of the following components:

Embeddings: LinSAKT processes input data that has question IDs, skill IDs and student responses. This data is transformed into dense vector representations called embeddings. The embeddings have the following components:

1. Question embedding: Each question is mapped to a dense vector of size d. These embeddings capture question related features.
2. Skill embedding: Every skill associated with a question is mapped to a dense vector of dimension d, representing the underlying KC.
3. Interaction embedding: This is a combination embedding involving student's response (correct: 1, incorrect: 0) concatenated with question and skill embedding. This embedding contextualizes the student response.

Causal Linear Attention: SAKT models a student's interaction history using a Transformer-style attention block. Given an input sequence representation $X \in R^{n \times d}$, each attention head forms linear projections $Q = XW^Q$, $K = XW^K$, $V = XW^V$, where q_t, $k_t \in R^d$, $v_t \in R^{d_v}$ denote the t-th rows of Q, K, V respectively. In the original scaled dot-product attention, the output is computed as:

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

Where Q, K, V are query, key and value matrices and d_k is the dimensionality of key (usually the same for query and value). This computation involves calculating a $n \times n$ matrix called the attention matrix. The order complexity of this matrix is $O(n^2)$, where n is the length of the sequence.
LinSAKT keeps the same SAKT preprocessing, projections, residual/FFN stack, and training procedure, and replaces only the SoftMax attention sublayer with a causal linear attention formulation that avoids constructing the full $n \times n$ attention matrix. We use a positive feature map φ(.) to approximate the attention kernel, specifically $\varphi(x) = elu(x) + 1$ applied elementwise. There are many such approaches given in literature, the approach used for developing LinSAKT is:

$$LinearAttention(Q, K, V) = \varphi(Q).(\varphi\left(K^T\right)V)$$

Here φ is a feature map (function) that allows a good approximation of attention. First the $(\varphi\left(K^T\right)V)$ calculation aggregates values weighted by keys. The result is multiplied by query specific weights $\varphi(Q)$. This shift in attention calculation reduces the complexity from $O(n^2)$ to $O(n)$ because pairwise calculations were avoided.

Causal Masking: In Attention calculation, context of a sequence is evaluated by considering the relationship between each element in a sequence. Causal attention only considers the context (contribution) of the previous elements in the sequence. This prevents 'future leakage' i.e. predictions cannot access future results in the training data. This falls in line with KT problems because a student's ability to answer a question depends on their prior interactions. To implement causal masking, consider a sequence of length n; for each timestep t, causality is enforced directly by aggregating only over the prefix $i \leq t$. Consider the running key-value matrix S_t and key vector z_t as:

$$S_t = \sum_{i=1}^{t} \phi\left(k_i\right) v_i^T \in R^{d \times d_v}, \quad z_t = \sum_{i=1}^{t} \phi\left(k_i\right) \in R^d,$$

$$o_t = \frac{\phi(q_t)^{\mathsf{T}} S_t}{\phi(q_t)^{\mathsf{T}} z_t + \varepsilon} \in R^{d_v}$$

This is equivalent to masked attention (no future leakage) but computed without forming the $n \times n$ attention matrix.

Implementation of LinSAKT: The causal linear attention described above utilizes embedding representation of student interactions to form a sequence. The process is outlined below:

1. Input: A sequence of interaction vectors called embeddings is defined as $X = (x_1, x_2, x_3 \ldots x_T)$, where each x_i is *d* dimensional and it combines question, skill and response information.
2. Query, Key Value Projections: The query (Q), key (K) and value (V) projections are calculated as $Q = XW^Q$, $K = XW^K$, $V = XW^V$, where W^Q, W^K, W^V are weight matrices $(d \times d)$ learned during the training.
3. Causal Linear Attention (CLA): For LinSAKT the feature map approximation used is $\varphi(x) = elu(x) + 1$, where *elu* is the exponential linear unit function. The aggregated value is calculated as $Z = \sum_{j=1}^{i} \varphi(k_j) v_j$ for each position i in the sequence. The attention output is calculated as $\varphi\left(q_i\right) . Z$.
4. Multi-Head Attention: LinSAKT uses multiple heads to capture various aspects of the input sequence. Each head computes a causal linear attention independently and the outputs are then concatenated through a linear layer.
5. Hyperparameters: LinSAKT model has 4 important hyperparameters that can be set to different values allowing for flexibility: Number of heads, Embedding/Model dimension *d*, Feature map φ (any positive feature map can be used) and sequence length *n*.

EXPERIMENTS AND RESULTS

In this section, the following research questions are answered:

RQ1: How does LinSAKT perform (runtime and accuracy) compared to the original SAKT (baseline) model?

RQ2: What is the impact of different model inputs on SAKT and LinSAKT model runtime and scalability (memory)?

Datasets and Evaluation Metrics To answer the research questions three publicly available datasets were used (url: https://sites.google.com/site/assistmentsdata/): 1) Assistment 2009-2010 data: Collected in the school year 2009-2010, this dataset is from skill builder (mastery learning - a free online tutoring platform developed by Worcester Polytechnic Institute (WPI)) problem sets. The dataset has 15931 students who solved at least 20 or more problems. 2) Assistment 2015 data: This has data from the 2015 school year with around 683801 student interactions contributed by 4151 students. The reason behind using these 2 datasets is to assess LinSAKT across two different dataset types when testing for problem-skill mapping i.e. Assistment 2009-2010 has one problem mapped to multiple skills while Assistment 2015 has one skill per problem. 3) Assistment Challenge data: This dataset was part of Assistment challenge in 2017. It has 942,816 interactions, 686 students with 102 skills. Of the three datasets this one has the greatest number of interactions which makes it a dense dataset.

We follow the standard SAKT preprocessing to convert each dataset into per-student interaction sequences. Offline, we (i) remove interactions with missing skill tags or missing correctness labels and drop duplicated records where applicable; (ii) map each unique skill tag to a contiguous integer index in $[0, n\_skill - 1]$; for ASSIST2009-2010, the corrected release collapses multi-skill tagging into a single combined tag (e.g., "skill1_skill2"), and we treat each combined tag as a distinct token; (iii) sort each student's interactions chronologically and export one row per student as two comma-separated sequences (item/skill IDs and binary correctness labels). The processed files are stored as tab-separated text with two columns. At training time, each sequence is truncated to the most recent n interactions (max_len = n) and right-padded to length n using padding IDs; we do not partition long histories into multiple fixed-length subsequences. Train/test splits are performed at the student level (one row per student) and stored as separate files (e.g., assist2015_train.csv and assist2015_test.csv) using an 80/20 split.

The problem is set up as a classification problem i.e. the model predicts the probability that a student will correctly answer a given question based on past interactions. Therefore, to evaluate the model the evaluation metrics used are: 1) Accuracy – This measures the ratio of correct predictions to the total number of predictions. 2) Precision – This metric gives information on correctly predicted positive observations. It is a ratio of true positives over all predictions that were predicted as positive. 3) Recall – This metric provides information on model's performance within the actual correct responses. It is a ratio of true positives over all predictions for positive observations. 4) F1 score – This is a harmonic mean of Precision and Recall and is used to balance the trade-off between the two metrics. It is useful in KT because sometimes data can be imbalanced or both false positives and false negatives can impact the outcome significantly.

Implementation Details

To demonstrate the effectiveness of LinSAKT for improving KT performance, the linear attention framework was integrated into a transformer-based architecture. We replicated the original SAKT experimental pipeline end-to-end with the modification of the attention sublayer. Each student sequence is represented as fixed-length tensors of item/skill IDs $q_1 .. q_n$ and interaction IDs $q_{a_1} .. q_{a_n}$, where $q_{a_t} = q_t + r_t * n\_skill$; correctness encoded as r_t in $\{0,1\}$ and n_skill unique skill ids. We use a start-of-sequence token ($2 * n\_skill$) and shift the interaction sequence by one step for next-step prediction: the model inputs are $[SOS,\ q_{a_1},\ ...,\ q_{a_{n-1}}]$ and queries are $q_1 .. q_n$. The model embeds q and q_a into d-dimensional vectors (question embedding and interaction embedding), adds learned positional embeddings, and applies a single attention block with 4 heads, followed by a position-wise feed-forward network with Relu and dropout 0.2, residual connections, and layer normalization. In the SAKT baseline, the attention sublayer is standard scaled dot-product attention; in LinSAKT, it is replaced with causal linear attention using an ELU(x) + 1 positive feature map while keeping all other components identical. The output layer produces a logit at each timestep, and we optimize masked binary cross-entropy over valid (non-padding) positions.

The model is implemented in PyTorch and executed on a single CUDA-capable GPU when available (Google Colab). Training uses the Adam optimizer (learning rate 1e-3) with binary cross-entropy loss (BCEWithLogits) and a StepLR learning-rate schedule (step_size = 1, gamma = 0.9). The attention output passes through a feed-forward network (hidden layer size 256, Relu activation, drop out of 0.2) with residual connections and sigmoid-activated prediction head. We train with batch size of 64; we use one random seed (seed = 42) per configuration (no repeated runs), and report test metrics after epoch 100; evaluation is performed on the test split using accuracy, precision, recall, and F1.

Model training and parameter selection

We use pre-generated train/test splits for each dataset (stored as separate files) and do not create an additional validation split. For each configuration, we set the embedding dimension *d* and maximum sequence length *n* directly through the model and dataset parameters (embed_dim = d, max_len = n). Each (*d, n*) configuration is trained for 100 epochs with identical optimization settings across SAKT and LinSAKT (Adam, learning rate 1e-3, no weight decay). Unless otherwise noted, results correspond to a single run per configuration; we report test metrics after the final epoch to avoid tuning on the test set.

Efficiency Reporting

For each dataset and each (*d, n*) setting, we compute percent reduction as $\Delta = (x_{SAKT} - x_{LinSAKT})/x_{SAKT}$, and report the overall average as the unweighted mean of Δover all datasets and all (d, n) configurations. In this formula, $x$denotes the performance metric being compared between SAKT and LinSAKT for a given dataset and a specific (n)configuration. Concretely, x is either the training time per epoch (measured in seconds) when reporting runtime improvement, or the peak GPU memory usage (measured in MB) when reporting memory improvement.

RQ1: SAKT and LinSAKT Comparison

Experiments were conducted using the three Assistment data sets. Detailed experimental results are presented in the Appendix. Table 1 reports best-value rows per dataset (not the overall average), which is why its memory reductions are larger (≈57% from ~790MB to ~340MB). Both models are comparable across accuracy, precision, recall and F1-score. For each dataset, a significant reduction in average runtime and maximum memory usage is observed for LinSAKT. This reduction varies across datasets due to the difference in the number of interactions and total number of users. Despite these differences LinSAKT performs consistently better than the SAKT model (Figure 1). In the best-value configurations summarized in Table 1, LinSAKT reduces runtime by 9.8%–27.2% and peak GPU memory by ~57% across the three datasets. For each dataset we observe the following reductions in memory and runtime respectively:

ASSIST2009 dataset: 790 to 340 MB (≈56.96%), 1.36 to 0.99 s (≈27.2%)
ASSIST2015 dataset: 790 to 341 MB (≈56.84%), 6.34 to 5.20 s (≈18.0%)
ASSIST2017 dataset: 790 to 340 MB (≈56.96%), 0.41 to 0.37 s (≈9.76%)

Table 1: Best-value configuration per dataset among the tested grid (d ∈ {64,128,256}, n ∈ {64,128,256,512}) (Bold indicates better performance per metric)

Dataset	**Model**	**AUC**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Average runtime (per epoch)**	**Peak GPU memory (MB) per epoch**
ASSIST2009	SAKT	0.74	0.73	0.74	0.89	0.81	1.36	790
	LinSAKT	**0.75**	0.73	0.74	0.89	0.81	**0.99**	**340**
ASSIST2015	SAKT	0.72	0.76	0.78	0.95	0.85	6.34	790
	LinSAKT	0.72	0.76	0.78	0.95	**0.86**	**5.20**	**341**
ASSIST2017	SAKT	0.61	0.64	0.65	0.93	0.76	0.41	790
	LinSAKT	**0.62**	0.64	0.65	**0.96**	0.76	**0.37**	**340**

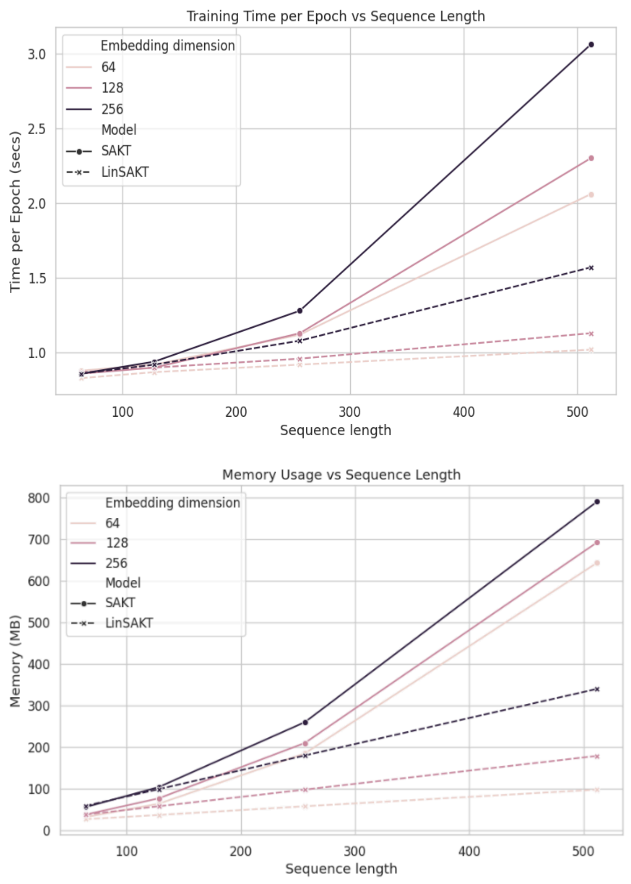
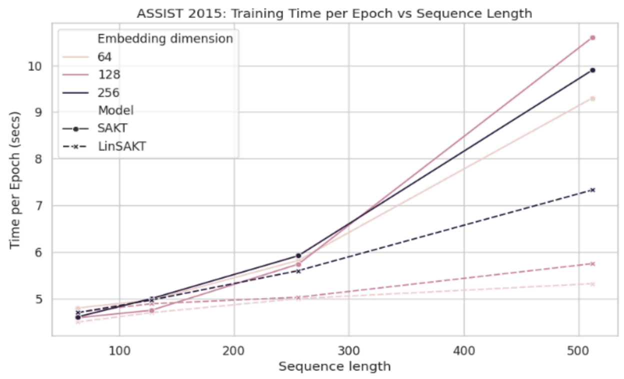


Figure 1: Training time and Memory usage for varying sequence length and embedding dimensions for ASSIST2009.

RQ2: What is the impact of different model inputs on SAKT and LinSAKT model runtime and scalability (memory)?

There are various hyperparameters (learning rate, batch size, dropout) that are used as inputs to the SAKT and LinSAKT model. Keeping these hyperparameters fixed, the goal of this analysis is to observe the impact of sequence length and embedding dimensions on time and memory usage of SAKT and LinSAKT. Figure 1 summarizes the results obtained across all experiments for ASSIST2009 dataset. For both the models, time increases almost linearly with sequence length. The time increase with embedding dimensions is slower. At higher sequence lengths, the time difference between the two models is significantly greater. For each model, the higher embedding dimensions lead to greater time per epoch. LinSAKT is consistently faster, especially with longer sequences, for example at sequence length 512 it is 2 times faster than SAKT. These results align with the theory behind attention mechanisms of both models: SAKT has quadratic complexity and LinSAKT has linear complexity. Trends from memory plot reveal that memory scales more with sequence lengths compared to embedding dimensions. For instance, doubling sequence length triples memory for SAKT model. This is due to storing of large attention matrices and all intermediate calculations plus activations. LinSAKT shows a similar trend, however when compared to SAKT it is much more efficient. For example, LinSAKT uses significantly less memory for higher embedding (256) and sequence lengths (512). These results reveal LinSAKT's optimizations for handling longer sequences without the need for quadratic matrices. The other two datasets (ASSIST2015 and ASSIST2017) confirm these trends (see Figure 2 and 3). In ASSIST2015, the training time reduction for LinSAKT (for a specific embedding dimension) is less than ASSIST2009. This could be attributed to the dramatic increase in the number of users from 4417 to 19917. In ASSIST2017, the time drop further reduces between the two models. One reason could be the reduced sparsity in the data set leading to SAKT performing better. However, memory reduction for LinSAKT across all datasets are highly significant. Because LinSAKT is more scalable and time efficient across different types of datasets, it is preferable for large-scale applications and/or experiments. Primary driver of time and memory usage reduction is sequence length. Because it offers marginal improvement in accuracy, sequences of smaller to moderate length should provide good results.



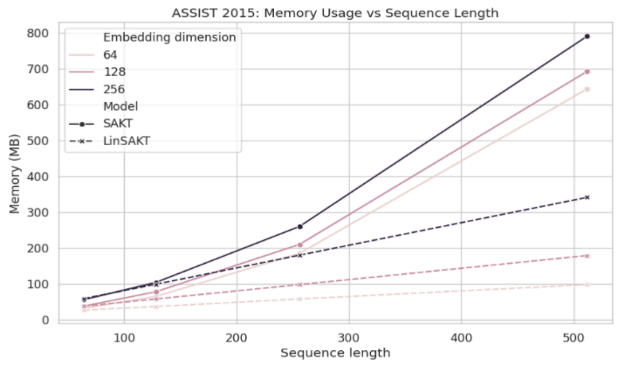
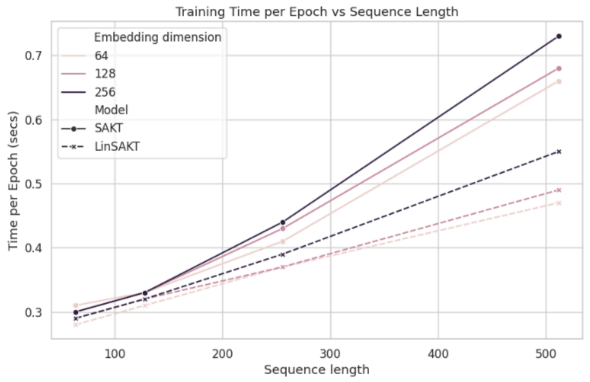


Figure 2: Training time and Memory usage for varying sequence length and embedding dimensions for ASSIST2015.



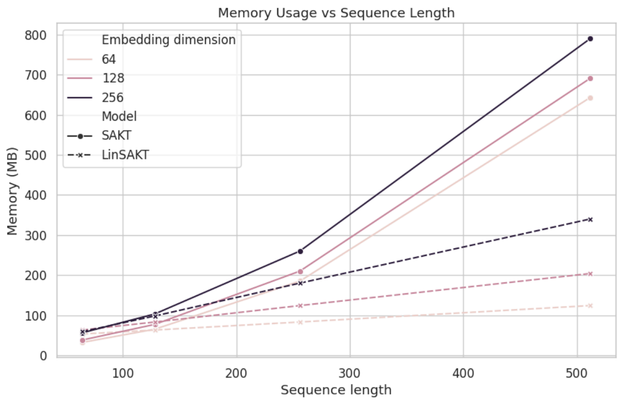


Figure 3: Training time and Memory usage for varying sequence length and embedding dimensions for ASSIST2017.

DISCUSSION

Attention-based frameworks are popular choice for KT problems due to their ability to model sequences better. SAKT is one such model that uses self-attention to model relevance between responses and exercises. This research introduced a new linear-attention-based framework called LinSAKT specifically for KT problems. An in-depth experimental analysis evaluated the performance and efficiency of both models over 3 different datasets: ASSIST2009, ASSIST2015 and ASSIST2017. Each model was tested for embedding dimensions of 64, 128 and 256 and sequence lengths of 64, 128, 256 and 512. Every configuration was run for 100 epochs. Results revealed a consistent pattern across datasets for each model offering insights into the trade-off between accuracy and model complexity. Both models achieved comparable predictive performance. The AUC value range for all datasets was small, and F1-scores did not vary much indicating a robust performance. Increasing embedding size or sequence lengths did not lead to larger predictive gains. This plateauing effect was most significant in ASSIST2015 most likely due to shorter interactions and less complex patterns in the dataset. When comparing efficiency, LinSAKT outperformed SAKT for every configuration. In ASSIST2009, LinSAKT reduced training time per epoch by up to 49% (for embedding 256, sequence 512) and memory usage up to 57% (340 MB versus 790MB). Similar trends were observed in the other two datasets. These dramatic reductions in runtime and memory usage are attributed to LinSAKT's linear attention mechanism (CLA), which mitigates the quadratic complexity of the self-attention in SAKT.

LinSAKT and SAKT give comparable performance across multiple datasets. This implies that for usage in edtech LinSAKT can be adopted given its advantage of reducing computational costs. For datasets with less complex patterns like ASSIST2015, high F1-scores indicate that LinSAKT can achieve robust KT and its high efficiency enables it to be deployed on low-resource devices like phones and tablets (often used in classrooms). For datasets like ASSIST2017, where low performance was achieved. One possible reason could be higher interaction complexity or noise levels because this dataset includes more recent and potentially diverse student interactions. LinSAKT's scalability can help in creating larger experiments with longer sequences and added features to improve accuracy. Another noticeable trend in all experiments is that with longer sequence lengths (above 128), there is a lack of significant improvement in performance. This could point to the fact that long-term dependencies might not be crucial to KT tasks as has been assumed in literature. For example, a student who was not performing well on some prior tasks might put in more effort at some point and start answering more questions correctly. Similarly, if someone was performing good on some easy to medium level skills they might start struggling in more difficult skills later. Incorporating more features in the form of student metadata like their level of preparation can lead to more accurate results.

Practically, LinSAKT is a candidate for real-time personalized learning systems where time and memory efficiency are critical. This application of LinSAKT falls in line with the growing demand for scalable AI in education, enabling applications in diverse settings like smaller rural schools or mobile-based learning platforms.

LIMITATIONS AND CONCLUSION

This study introduced a novel KT model called LinSAKT for improved efficiency and lower memory usage. There are a few limitations around the experimental study of this model. Only ASSIST datasets were used. These datasets might not capture diverse nature of student interactions in other types of learning environments. The clustering of performance metrics around the same range is pointing to this limitation. In future work we aim to analyze other types of datasets to further study benefits of reduced model complexity. All experiments were conducted on the same hardware using the same setup (Google Colab). It might be useful to capture the scaling benefits of LinSAKT under different hardware settings. Finally, we did not explore systematic hyperparameter tuning for all input variables because the focus was to study the impact of sequence length and embedding dimensions. In future studies our goal is to explore this aspect. Additionally, benchmarking LinSAKT's performance and efficiency across varied hardware (devices, GPUs, etc.) will clarify its applicability in real-world. We report runtime and peak GPU memory, but we do not instrument power draw or estimate CO_2; evaluating energy/carbon impact under specific hardware and grid assumptions is left to future work.

LinSAKT is a first step towards democratization of AI-driven education tools. This model reduces training time and saves significantly on memory; therefore, it can enable scalable knowledge tracing in low-resource environments. With improved efficiency, this model can support equitable access to personalized learning tools in underserved regions. This model contributes to the growing field of efficient AI and supports efforts to develop sustainable and scalable educational technologies. LinSAKT offers a blueprint for designing AI systems for education that are effective and scalable.

REFERENCES

Abdelrahman, G., Wang, Q., & Nunes, B. (2023). Knowledge tracing: A survey. *ACM Computing Surveys, 55*(11), 1–37. https://doi.org/10.1145/3569576

Ansar, W., Goswami, S., & Chakrabarti, A. (2024). A survey on transformers in NLP with focus on efficiency. *arXiv preprint arXiv:2406.16893*. https://arxiv.org/abs/2406.16893

Bannour, N., et al. (2021). Evaluating the carbon footprint of NLP methods: A survey and analysis of existing tools. In *Proceedings of the second workshop on simple and efficient natural language processing* (pp. 11–21). Association for Computational Linguistics.

Brown, T., et al. (2020). Language models are few-shot learners. In *Advances in neural information processing systems 33* (pp. 1877–1901). Neural Information Processing Systems Foundation.

Devlin, J., et al. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 4171–4186). Association for Computational Linguistics.

Li, H., et al. (2021). One-shot learning with memory-augmented neural networks using a 64-kbit, 118 GOPS/W RRAM-based non-volatile associative memory. In *2021 Symposium on VLSI Technology* (pp. 1–2). IEEE. https://doi.org/10.23919/VLSITechnology47932.2021.9471057

Liu, L., et al. (2023). Linrec: Linear attention mechanism for long-term sequential recommender systems. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 184–193). ACM. https://doi.org/10.1145/3539618.3591703

Mandalapu, V., Gong, J., & Chen, L. (2021). Do we need to go deep? Knowledge tracing with big data. *arXiv preprint arXiv:2101.08349*. https://arxiv.org/abs/2101.08349

Pandey, S., & Karypis, G. (2019). A self-attentive model for knowledge tracing. *arXiv preprint arXiv:1907.06837*. https://arxiv.org/abs/1907.06837

Piech, C., et al. (2015). Deep knowledge tracing. In *Advances in neural information processing systems 28* (pp. 505–513). Neural Information Processing Systems Foundation.

Qin, Z., et al. (2022). Cosformer: Rethinking softmax in attention. *arXiv preprint arXiv:2202.08791*. https://arxiv.org/abs/2202.08791

Shen, S., et al. (2024). A survey of knowledge tracing: Models, variants, and applications. *IEEE Transactions on Learning Technologies, 17*, 1858–1879. https://doi.org/10.1109/TLT.2024.3383327

Vaswani, A., et al. (2017). Attention is all you need. In *Advances in neural information processing systems 30* (pp. 5998–6008). Neural Information Processing Systems Foundation.

Wang, H., et al. (2016). Machine learning basics. In *Deep learning* (pp. 98–164). MIT Press.

Wang, S., et al. (2020). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*. https://arxiv.org/abs/2006.04768

Yeung, C.-K., & Yeung, D.-Y. (2018). Addressing two problems in deep knowledge tracing via prediction-consistent regularization. In *Proceedings of the fifth annual ACM conference on learning at scale* (Article No. 15). ACM. https://doi.org/10.1145/3231644.3231647
Zaheer, M., et al. (2020). Big bird: Transformers for longer sequences. In *Advances in neural information processing systems 33* (pp. 17283–17297). Neural Information Processing Systems Foundation.
Zheng, L., Wang, C., & Kong, L. (2022). Linear complexity randomized self-attention mechanism. In *International conference on machine learning* (pp. 27041–27061). PMLR.
Liang, Z., Wu, R., Liang, Z., Yang, J., Wang, L., & Su, J. *GELT: A graph embeddings based lite-transformer for knowledge tracing.* PLOS ONE, 19(5), e0301714 (2024). https://doi.org/10.1371/journal.pone.0301714
Yao, S., Song, Y., Liu, Y., Chen, J., Zhao, D., & Wang, X. *ANT-KT: Adaptive NAS Transformers for Knowledge Tracing.* **Electronics**, 14(21), 4148 (2025). https://doi.org/10.3390/electronics14214148

APPENDIX

Code: https://github.com/IzhanAli08/Knowledge_Tracing
Results:
Table 2: ASSIST2017 Results: Each run for 100 epochs

Model	Embedding dimension	Sequence length	AUC	Accuracy	Precision	Recall	F1-score	Time/ epoch (in secs)	Memory (MB)
SAKT	64	64	0.57	0.60	0.61	0.89	0.73	0.31	32

	64	128	0.57	0.62	0.63	0.90	0.74	0.33	65						
	64	256	0.59	0.63	0.64	0.92	0.75	0.41	185						
	64	512	0.59	0.64	0.64	0.93	0.76	0.66	643						
	128	64	0.60	0.62	0.63	0.85	0.73	0.30	38						
	128	128	0.58	0.62	0.64	0.87	0.74	0.33	77						
	128	256	0.60	0.63	0.65	0.89	0.75	0.43	210						
	128	512	0.62	0.64	0.65	0.90	0.76	0.68	691						
	256	64	0.58	0.60	0.63	0.81	0.71	0.30	56						
	256	128	0.58	0.61	0.64	0.85	0.73	0.33	103						
	256	256	0.60	0.63	0.65	0.86	0.74	0.44	260						
	256	512	0.61	0.64	0.65	0.89	0.75	0.73	790						
LinSAKT	64	64	0.57	0.61	0.61	0.93	0.74	0.28	53						
	64	128	0.58	0.62	0.63	0.93	0.75	0.31	63						
	64	256	0.59	0.63	0.64	0.95	0.76	0.37	83						
	64	512	0.59	0.63	0.64	0.96	0.76	0.47	124						
	128	64	0.59	0.62	0.64	0.87	0.74	0.29	63						
	128	128	0.59	0.63	0.63	0.90	0.75	0.32	83						
	128	256	0.60	0.64	0.64	0.92	0.76	0.37	124						
	128	512	0.61	0.64	0.65	0.93	0.76	0.49	204						
	256	64	0.60	0.62	0.64	0.87	0.74	0.29	59						

	256	128	0.61	0.63	0.64	0.89	0.75	0.32	98					
	256	256	0.60	0.64	0.65	0.90	0.76	0.39	180					
	256	512	0.62	0.64	0.65	0.90	0.76	0.55	340					

Table 3: ASSIST2009 Results: Each run for 100 epochs

Model	**Embedding dimension**	**Sequence length**	**AUC**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Time/epoch (in secs)**	**Memory (MB)**
SAKT	64	64	0.74	0.70	0.72	0.86	0.79	0.88	32
	64	128	0.74	0.71	0.72	0.87	0.79	0.92	64
	64	256	0.74	0.71	0.73	0.89	0.80	1.12	185
	64	512	0.74	0.72	0.74	0.89	0.81	2.06	643
	128	64	0.74	0.71	0.72	0.86	0.79	0.86	38
	128	128	0.74	0.71	0.73	0.87	0.79	0.90	77
	128	256	0.74	0.72	0.73	0.88	0.80	1.13	210
	128	512	0.74	0.72	0.74	0.89	0.81	2.30	692
	256	64	0.74	0.71	0.72	0.86	0.78	0.86	56
	256	128	0.73	0.71	0.72	0.87	0.79	0.94	104
	256	256	0.74	0.71	0.73	0.88	0.80	1.28	260
	256	512	0.74	0.73	0.74	0.89	0.81	3.06	790
LinSAKT	64	64	0.74	0.71	0.72	0.86	0.79	0.83	27
	64	128	0.74	0.71	0.73	0.87	0.79	0.87	37
	64	256	0.74	0.72	0.73	0.89	0.80	0.92	58
	64	512	0.74	0.72	0.74	0.89	0.81	1.02	98
	128	64	0.75	0.71	0.73	0.86	0.79	0.87	38
	128	128	0.75	0.71	0.73	0.87	0.79	0.90	58
	128	256	0.74	0.72	0.73	0.89	0.80	0.96	98
	128	512	0.74	0.72	0.74	0.89	0.81	1.13	179
	256	64	0.75	0.72	0.73	0.86	0.79	0.86	59
	256	128	0.74	0.72	0.74	0.87	0.80	0.92	99
	256	256	0.75	0.72	0.74	0.89	0.80	1.08	180
	256	512	0.74	0.73	0.74	0.89	0.81	1.57	340

Table 4: ASSIST2015 Results: Each run for 100 epochs

Model		**Embedding dimension**	**Sequence length**	**AUC**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Time/epoch (in secs)**	**Memory (MB)**
	SAKT	64	64	0.72	0.76	0.78	0.95	0.85	4.80	32
		64	128	0.72	0.76	0.77	0.95	0.85	4.97	65
		64	256	0.72	0.75	0.77	0.95	0.85	5.82	185
		64	512	0.72	0.75	0.77	0.95	0.85	9.30	643
		128	64	0.71	0.76	0.78	0.94	0.85	4.59	38
		128	128	0.71	0.75	0.77	0.94	0.85	4.75	78
		128	256	0.72	0.75	0.77	0.94	0.85	5.74	210
		128	512	0.71	0.75	0.77	0.94	0.85	10.6	692
		256	64	0.71	0.75	0.78	0.93	0.85	4.61	56
		256	128	0.71	0.75	0.78	0.93	0.85	5.00	104
		256	256	0.71	0.75	0.78	0.93	0.85	5.92	260
		256	512	0.72	0.75	0.78	0.93	0.85	9.90	790
	LinSAKT	64	64	0.72	0.76	0.78	0.95	0.85	4.50	27
		64	128	0.72	0.75	0.77	0.95	0.85	4.70	37
		64	256	0.71	0.75	0.77	0.95	0.85	5.00	58
		64	512	0.72	0.75	0.77	0.95	0.85	5.32	98
		128	64	0.72	0.76	0.78	0.94	0.85	4.71	38
		128	128	0.72	0.76	0.77	0.95	0.85	4.89	58
		128	256	0.72	0.75	0.77	0.94	0.85	5.03	98
		128	512	0.72	0.76	0.77	0.95	0.86	5.75	179
		256	64	0.72	0.76	0.77	0.95	0.86	4.70	59
		256	128	0.72	0.76	0.77	0.95	0.86	4.97	99
		256	256	0.72	0.76	0.76	0.94	0.86	5.6	180
		256	512	0.72	0.75	0.77	0.95	0.86	7.33	341