Deep Knowledge Tracing for Explainable Problem Recommendations on Codeforces

James Zhao[*] , **Fang Sun**[*] and **Yizhou Sun**
UCLA ([*] Equal contribution)
jdzhao@g.ucla.edu, {fts, yzsun}@cs.ucla.edu

Abstract

Contemporary competitive programming platforms, such as Codeforces, offer a vast array of problems, which can overwhelm novice users encountering competitive programming for the first time. This complexity highlights the need for intelligent recommendation learning paths. However, designing these paths using traditional recommendation systems poses significant challenges, as many do not incorporate temporal knowledge and prerequisite concepts, often relying solely on correlation-based methods. We approach this issue from the perspective of deep knowledge tracing (DKT), utilizing transformers for predicting users' skill levels and recommending problems. Our model employs DKT to learn a dynamic knowledge vector that predicts the probability of users successfully solving any given problem. Additionally, we enhance our DKT transformer architecture with a TransE-based prerequisite graph. Our model achieves a ROC AUC score of 0.75 for knowledge tracing, paving the way for explainable recommendations. Users who interact with our system benefit from real-time insights into their weaknesses while receiving targeted suggestions to improve their knowledge.

1 Introduction

The online competitive programming platform Codeforces contains a broad array of over 30,000 programming problems with an exceptionally high skill ceiling, making it a gold mine for a maximal level of human skill development that even state-of-the-art AI models are currently unable to reach [Li *et al.*, 2022]. However, navigating this collection of programming problems is particularly tricky for the novice. Even with problem tags and difficulty levels, new users can be bogged down due to concept unfamiliarity and the asymmetry of their own skills. In general, unstructured exploration of such an educational resource has been shown to be inferior to structured guidance [Kirschner *et al.*, 2006]. Thus, there exists the need for an automated learning system that can understand both the user's skill level (measured in terms of a "knowledge frontier") and the structural dependencies of the learning resource to create an optimal "golden path" that challenges but does not overwhelm the user [da Silva *et al.*, 2023].

Traditional recommendation algorithms, while successful in domains like e-commerce [Sarwar *et al.*, 2001], often rely on collaborative filtering or content-based methods that capture correlational patterns [Resnick *et al.*, 1994; Goldberg *et al.*, 1992]. These approaches may not adequately model the hierarchical nature of knowledge acquisition or the prerequisite dependencies inherent in educational subjects like algorithmic problem-solving. For instance, understanding graph traversal is typically a prerequisite for tackling complex shortest path algorithms. A purely correlation-based system might miss this "you should master A before attempting B" nuance, leading to recommendations that are either too trivial or prematurely advanced, thereby hindering optimal learning.

Causal learning aims to model underlying cause-effect relationships, offering the potential for more reliable, fair, and interpretable systems [Kumar *et al.*, 2023]. In an educational context, this involves two primary aspects:

1. **Causal Discovery (Knowledge Structure):** Identifying dependencies between concepts or problems, often represented as a prerequisite graph where mastering one skill or problem enables or facilitates the understanding of another [Roy *et al.*, 2019].
2. **Causal Inference (Learning Impact):** Estimating the effect of a pedagogical intervention, such as attempting a specific problem, on a learner's knowledge state and their ability to solve future problems.

Our work focuses on leveraging the temporal sequence of user interactions on Codeforces to address the causal inference aspect directly through Deep Knowledge Tracing (DKT). By modeling how a user's knowledge state evolves with each problem solved or failed, we can predict their current capability across all problems. This "knowledge vector", a set of probabilities indicating the likelihood of solving each problem, forms the basis for our recommender. Such a system can offer explanations grounded in the user's learning trajectory and the predicted impact of new problems on their knowledge. For instance, a problem might be recommended because the user has a high probability of solving it and because solving it is predicted to significantly improve their chances on other, more advanced problems (i.e., maxi-

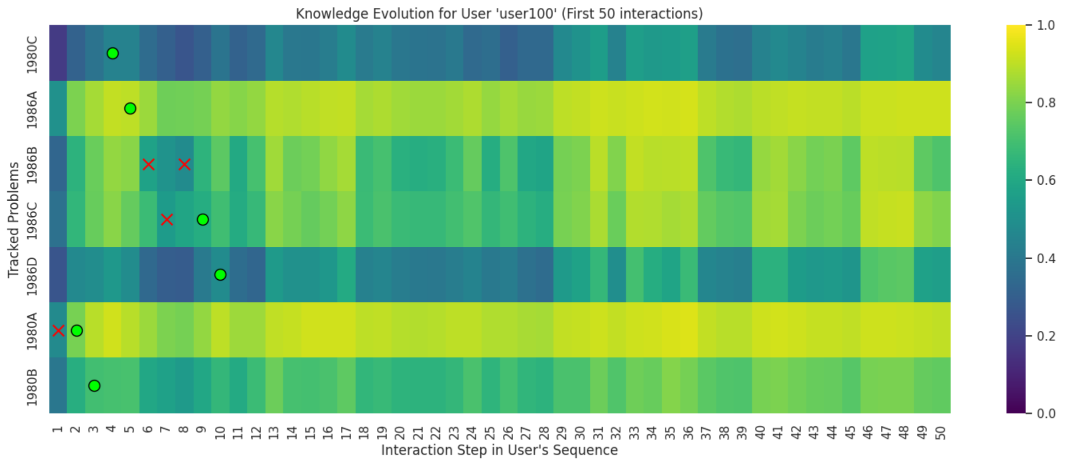

Figure 1: Figure depicting the model's predicted solve probabilities for seven problems (y-axis) versus fifty steps in the user's problem-solving sequence. At the first step, the model is not confident the user can solve anything. The first ten steps, marked by X's (incorrect attempts) and circles (correct submissions) are shown. The model is able to update is estimate of the user's solve probability for each problem after each attempt by feeding the new interaction I_t into the transformer model.

mizing their expected knowledge gain).

Developing an explicit prerequisite graph from data as part of causal discovery holds significant value. We investigate using embeddings derived from this graph as features, based on which we propose a DKT system designed to inherently follow knowledge. This approach enables us to: (1) Predict a user's dynamic, fine-grained knowledge state; (2) Formulate recommendations that aim to expand the user's "knowledge frontier" by suggesting challenging yet achievable problems; (3) Provide explanations for recommendations based on predicted solvability and potential learning impact.

We utilize a large-scale dataset of Codeforces submissions to train and evaluate our DKT Transformer model. Our preliminary results show that the DKT model effectively traces user knowledge (validation AUC ≈ 0.75). This work paves the way for more personalized and pedagogically sound guidance on competitive programming platforms.

The major contributions of our model can be summarized as follows:

- **Causal DKT–Transformer.** We treat each problem attempt as a causal intervention on the learner's latent state, and use a transformer-based Deep Knowledge Tracing model to produce a dynamic "knowledge vector" of success probabilities over all problems.
- **Data-driven prerequisite graph.** We recover causal dependencies between problems from Codeforces submission logs, embed them via TransE, and integrate these embeddings to bias the DKT representation toward the true hierarchy of concept prerequisites.
- **Intervention-aware evaluation.** We introduce a temporal-holdout protocol combining ROC AUC for tracing fidelity with a HITS@K metric that measures how recommended "interventions" (problems) causally improve downstream solving performance.
- **Explainable, impact-driven recommendations.** We show how our causally-grounded tracing yields transparent suggestions—each recommendation comes with both a predicted solvability score and an estimate of its expected knowledge gain.

2 Related Work

Our research intersects with several areas: knowledge tracing, sequential recommendation, graph-based recommendation, causal inference in education, and recommender systems for programming education.

Knowledge Tracing (KT). KT aims to model a student's understanding of concepts over time based on their interactions with educational content [Corbett and Anderson, 1994]. Early models like Bayesian Knowledge Tracing (BKT) often required explicit mapping of problems to underlying skills. Deep Knowledge Tracing (DKT) [Piech *et al.*, 2015] marked a significant advancement by using Recurrent Neural Networks (RNNs) to automatically learn latent knowledge states from sequences of problem interactions (problem ID and correctness). This removed the need for explicit skill tagging for each problem, allowing models to learn complex temporal dependencies.

Subsequent work has extended DKT using more powerful sequential architectures. Attention mechanisms, popularized by the Transformer [Vaswani *et al.*, 2017], have been incorporated into KT models like Self-Attentive Knowledge Tracing (SAKT) [Pandey and Karypis,] and SAINT(+) [Shin *et al.*, 2021], often outperforming RNN-based DKTs by better capturing long-range dependencies in student learning histories. These models typically predict the probability of a correct response on the next interaction. Our approach builds on this line of work, employing a Transformer architecture to process rich interaction sequences from Codeforces, including problem difficulty, user ratings, and time deltas, to predict solvability across the entire problem set.

Sequential Recommendation. Parallel to KT, sequential recommendation systems also model sequences of user-item interactions, but typically aim to predict the next item a user will interact with (e.g., click, purchase, watch) [Hidasi *et al.*, 2016; Rendle *et al.*, 2010]. Transformer-based models like SASRec [Kang and McAuley, 2018] and BERT4Rec [Sun *et al.*, 2019] have shown state-of-the-art performance. While the modeling techniques are similar to those in Transformer-based KT, the prediction task differs: KT focuses on predicting correctness/mastery, while sequential recommenders predict item choice. Our DKT model's output (probabilities of solving all problems) can be seen as a rich user state representation, which could inform a subsequent next-item prediction layer, but our primary goal is assessing solvability for learning-focused recommendations.

Causal Reasoning and Explainability in Recommender Systems. There's a growing interest in moving beyond correlation-based recommendations towards systems that understand or infer causal relationships [Wang *et al.*, 2023; Zhang *et al.*, 2021]. This is particularly vital in education, where the effect of working on a problem on future learning is key. Causal approaches address issues like selection bias [Schnabel *et al.*, 2016] and aim to provide more robust and fair recommendations. Explainable AI (XAI) for recommenders also seeks to make system outputs transparent [Zhang and Chen, 2020]. Our DKT approach contributes to explainability by grounding recommendations in predicted solvability and potential "knowledge gain" derived from the model's simulation of solving a problem, offering a "why" behind suggestions. While true causal claims from observational data are challenging, DKT allows for causal-style interpretations of how specific interactions impact the learned knowledge state.

Prerequisite Modeling and Graph-Based Recommendation. Explicitly modeling prerequisites is a form of incorporating causal domain structure. This often involves constructing a prerequisite graph where nodes are concepts or problems and edges denote dependencies [Chen *et al.*, 2018; Roy *et al.*, 2019]. Knowledge Graph (KG) embedding techniques like TransE [Bordes *et al.*, 2013], TransH [Wang *et al.*, 2014], etc., learn vector representations of entities and relations in such graphs. These embeddings can then enrich recommendation models [Wang *et al.*, 2019]. Graph Neural Networks (GNNs) also directly learn from graph structures [Wu *et al.*, 2022; Ying *et al.*, 2018]. In our current DKT-focused work, we propose using these pre-computed TransE embeddings as additional features to the DKT model, allowing it to leverage this explicit structural information alongside its sequence-based learning.

Recommendation in Competitive Programming. Several systems aim to recommend problems on platforms like Codeforces, often using collaborative filtering, problem tags, or difficulty [Desai, 2023; Coc, 2021]. Shared datasets [Denk, 2024] have spurred research. However, few systems deeply integrate temporal knowledge tracing with the goal of providing explainable, learning-focused paths sensitive to prerequisite structures learned from data. Our work aims to advance this by applying a robust DKT Transformer model, potentially enhanced by explicit prerequisite graph embeddings, to provide fine-grained solvability predictions for personalized learning trajectories.

3 Proposed Methodology

In this section, we describe our end-to-end methodology for causal, explainable problem recommendation on Codeforces. We begin by detailing how we curate and transform raw submission logs into rich sequential interaction data, including problem identifiers, outcomes, difficulty ratings, user ratings, and interaction time deltas, and split them into user-level temporal training and test sets. We then introduce our DKT-Transformer architecture, which ingests these engineered features, applies causal self-attention to model the effect of each problem attempt on the learner's latent knowledge state, and outputs a dynamic "knowledge vector" of solve probabilities. To infuse explicit prerequisite structure, we explain how we derive a directed acyclic problem graph via an LIS-inspired analysis of solving patterns, embed its edges with TransE, and seamlessly integrate these embeddings into the DKT input representation. Finally, we outline our intervention-aware recommendation strategy, which leverages the resulting knowledge vector to select problems that the learner is likely to solve and whose predicted causal impact maximizes expected knowledge gain.

3.1 Dataset and Feature Engineering

We utilize a large-scale dataset derived from Codeforces submission logs, based on the "UsersCodeforcesSubmissionsEnd2024" dataset by Denk [Denk, 2024]. Each raw interaction record contains the user's handle, problem ID, submission timestamp, the problem's official rating, the user's Codeforces rating at the time of submission, and the verdict of the submission.

Core Interaction Definition. For our DKT model, each step in a user's sequence represents a problem interaction. The primary features for each interaction at step t are:

- **Problem ID (P_t):** The unique identifier of the problem attempted, label-encoded into a continuous integer range.
- **Outcome (O_t):** A binary variable indicating success (1 if verdict is 'OK', 0 otherwise for relevant non-successful attempts like 'WRONG_ANSWER', 'TIME_LIMIT_EXCEEDED').
- **Problem Rating (PR_t):** The official difficulty rating of P_t.
- **User Rating at Submission (UR_t):** The user's overall Codeforces rating when attempting P_t.
- **Time Delta (ΔT_t):** The time elapsed since the user's previous interaction ($T_t - T_{t-1}$). For the first interaction, $\Delta T_1 = 0$.

Continuous features ($PR_t, UR_t, \Delta T_t$) are normalized (e.g., using standardization based on training set statistics, with ΔT_t log-transformed before standardization due to its skewed distribution). Missing values for ratings are imputed using medians derived from the training set.

Sequence Generation and Splitting. User interactions are first sorted chronologically for each unique user to form individual interaction histories. From these histories, we construct sequences suitable for our DKT model.

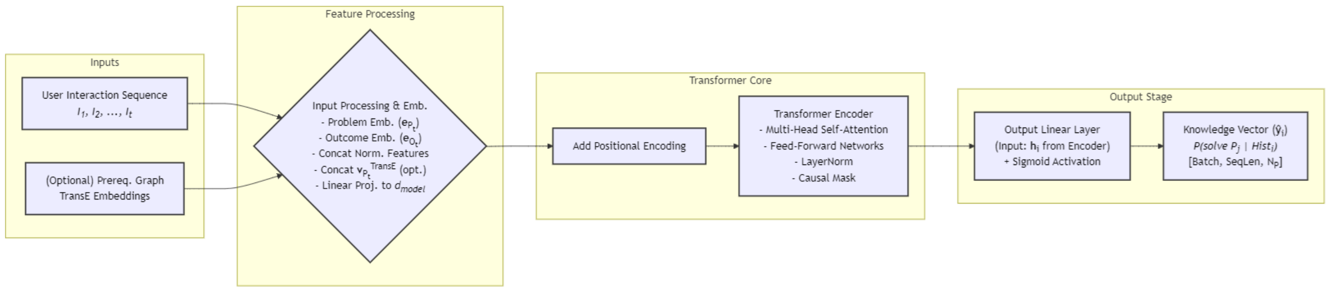
Inputs
User Interaction Sequence $I_1, I_2, \ldots, I_t$
(Optional) Prereq. Graph TransE Embeddings
Feature Processing
Input Processing & Emb.
- Problem Emb. (e_{P_t})
- Outcome Emb. (e_{O_t})
- Concat Norm. Features
- Concat $v_{P_t}^{TransE}$ (opt.)
- Linear Proj. to d_{model}
Transformer Core
Add Positional Encoding
Transformer Encoder
- Multi-Head Self-Attention
- Feed-Forward Networks
- LayerNorm
- Causal Mask
Output Stage
Output Linear Layer
(Input: $\mathbf{h}_i$ from Encoder)
+ Sigmoid Activation
Knowledge Vector ($\hat{y}_i$)
$P(solve\ P_j \mid Hist_i)$
[Batch, SeqLen, N_P]

Figure 2: Conceptual overview of the DKT-Transformer model, taking sequential user interactions and prerequisite graph embeddings (optional) as input to predict a knowledge vector (solve probabilities).

Each sequence consists of a series of interaction tuples $(P_t, O_t, PR_t, UR_t, \Delta T_t)$ as defined previously. For training the DKT model to predict the outcome of the $(k+1)^{th}$ interaction based on the preceding k interactions, we transform each user's history of length L into input sequences of features from interactions $(I_0, \ldots, I_{L-2})$ and corresponding target sequences of problem IDs $(P_1, \ldots, P_{L-1})$ and their actual outcomes $(O_1, \ldots, O_{L-1})$.

User sequences with fewer than two interactions are filtered out, as they do not provide a basis for next-step prediction. To manage computational complexity and ensure uniform input dimensions for the Transformer, sequences longer than a predefined 'MAX_SEQ_LENGTH' are truncated, retaining the most recent interactions. Shorter sequences are padded to this 'MAX_SEQ_LENGTH' using specific padding values for problem IDs, outcomes, and continuous features. Crucially, these processed sequences are then subjected to a strict *user-level temporal split*: users are divided into training (80%) and testing (20%) sets, ensuring that all sequences from a given user belong exclusively to one set. This setup rigorously tests the model's ability to generalize to entirely new users.

3.2 DKT Transformer Model Architecture

We adapt the Transformer architecture [Vaswani *et al.*, 2017] for the DKT task. The model processes a sequence of past interactions to predict the probability of solving each problem in the entire problem set.

Input Embedding Layer. For each interaction $I_t = (P_t, O_t, PR_t, UR_t, \Delta T_t)$ in a user's sequence, we construct a rich input representation:

1. **Problem Embedding ($\mathbf{e}_{P_t}$):** A learned embedding vector for P_t.
2. **Outcome Embedding ($\mathbf{e}_{O_t}$):** A learned embedding vector for the binary outcome O_t.
3. **Continuous Features:** The normalized $PR_t, UR_t, \Delta T_t$ are treated as scalar features.

These components are concatenated:

$$\mathbf{x}_t = [\mathbf{e}_{P_t}; \mathbf{e}_{O_t}; PR_t; UR_t; \Delta T_t] \tag{1}$$

This combined vector $\mathbf{x}_t$ is then projected by a linear layer to the Transformer's hidden dimension, d_{model}.

Positional Encoding. Standard sinusoidal positional encodings are added to the projected interaction embeddings to provide the model with information about the order of interactions in the sequence.

Transformer Encoder. The sequence of positionally-aware interaction embeddings is fed into a multi-layer Transformer encoder. Each encoder layer consists of a multi-head self-attention mechanism followed by a position-wise feed-forward network. Layer normalization and dropout are applied within each layer. The self-attention mechanism allows the model to weigh the importance of all prior interactions when constructing the representation for the current state. A causal (triangular) attention mask is applied to ensure that the prediction for step i only depends on interactions $1, \ldots, i$.

Output Layer. After processing the input sequence of length L_{in} (representing interactions $I_1, \ldots, I_{L_{in}}$), the Transformer encoder outputs a sequence of context vectors $\mathbf{h}_1, \ldots, \mathbf{h}_{L_{in}}$, where $\mathbf{h}_i \in R^{d_{model}}$ is the representation of the user's knowledge state after the i-th interaction in the input. For each $\mathbf{h}_i$, a final linear layer followed by a sigmoid activation function predicts the probability of solving every problem P_j in the entire problem set (of size N_P):

$$\hat{\mathbf{y}}_i = \sigma(W_{out}\mathbf{h}_i + \mathbf{b}_{out}) \in [0,1]^{N_P} \tag{2}$$

Thus, $\hat{\mathbf{y}}_i[j]$ is the predicted probability $P(\text{user solves problem } P_j | \text{History up to } I_i)$.

Training Objective. The model is trained to predict the outcome of the *next actual interaction*. If the input sequence to the DKT model at a given training step consists of interactions $I_1, \ldots, I_k$, and the user's actual $(k+1)^{th}$ interaction was with problem P_{tgt} resulting in outcome O_{tgt}, the loss is computed based on the model's prediction for P_{tgt} made from state $\mathbf{h}_k$. Specifically, we use Binary Cross-Entropy (BCE) loss:

$$\begin{aligned}\mathcal{L} = -[&O_{tgt} \log(\hat{\mathbf{y}}_k[P_{tgt}]) \\ &+ (1 - O_{tgt}) \log(1 - \hat{\mathbf{y}}_k[P_{tgt}])]\end{aligned} \tag{3}$$

The loss is calculated only for valid (non-padded) target interactions in a batch.

3.3 Enhancing DKT with Prerequisite Graph Embeddings

To provide the DKT model with explicit knowledge about problem dependencies, we leverage embeddings from a problem prerequisite graph described in Section 3.4 to inform our DKT model.

Prerequisite Graph Construction and TransE Embeddings. In Section 3.4 we describe the process of constructing a directed acyclic graph where an edge $A \to B$ indicates that problem A is a prerequisite for problem B. This graph is derived from temporal user solving patterns and problem difficulty ordering, using a longest increasing subsequences (LIS)-inspired approach. We then apply TransE [Bordes *et al.*, 2013] to this graph to learn a d_{emb}-dimensional embedding $\mathbf{v}_P^{TransE}$ for each problem P, such that if $A \to B$, then

$$\mathbf{v}_A^{TransE} + \mathbf{r} \approx \mathbf{v}_B^{TransE}, \tag{4}$$

where $\mathbf{r}$ is the embedding for the "isPrerequisiteOf" relation.

Integration into DKT. These pre-trained TransE embeddings are incorporated into the DKT model's input representation. When constructing the feature vector $\mathbf{x}_t$ for an interaction I_t involving problem P_t:

$$\mathbf{x}_t = [\mathbf{e}_{P_t}; \mathbf{e}_{O_t}; \mathbf{v}_{P_t}^{TransE}; PR_t; UR_t; \Delta T_t] \tag{5}$$

The TransE embedding $\mathbf{v}_{P_t}^{TransE}$ is concatenated with the other features. The DKT model's own problem embedding layer for $\mathbf{e}_{P_t}$ can either be initialized randomly or potentially with these TransE embeddings (and then fine-tuned). An ablation study (Section 4) will assess the impact of including these TransE features.

3.4 LIS-based Prerequisite Graph Generation

We infer prerequisite relationships between Codeforces problems by analyzing users' temporal solving patterns, with a focus on sequences of increasing difficulty. The intuition is that if users consistently solve an easier problem A before successfully tackling a harder problem B, then A is likely a prerequisite for B. The algorithm aims to be efficient by processing each user's solved problem sequence in a single pass, inspired by Longest Increasing Subsequence (LIS) construction.

Problem Difficulty Ordering and Preprocessing

First, all unique problems in the dataset successfully solved by users (verdict == `OK`) are considered. For each such problem, we determine its median problem rating based on all successful submissions. Problems are then globally sorted by this median rating to establish a canonical order of difficulty. Let $P'_0, P'_1, \ldots, P'_{N_P-1}$ be the sequence of problems sorted by increasing difficulty, where N_P is the total number of unique problems after an initial pruning step (e.g., removing problems solved by fewer than a minimum number of users, such as 4). Each problem P is mapped to its global difficulty index $\text{idx}(P)$ based on this sorted list.

Per-User LIS-Inspired Path Processing

For each user, their sequence of successfully solved problems is processed chronologically. During this process, we maintain a dynamic list, `difficulty_seq`, which stores the difficulty indices of problems forming the current LIS of problem difficulties encountered so far for that user.

When processing a newly solved problem P_{current} (with difficulty index $\text{idx}(P_{\text{current}})$) by the user:

1. If P_{current} is not in our pre-filtered and indexed set of problems, or if P_{current} has already been processed in the user's current LIS path construction (to handle first solves primarily), it is skipped.
2. We find the position *pos* where $\text{idx}(P_{\text{current}})$ would be inserted into the current `difficulty_seq` to maintain its sorted order (using binary search, e.g., `bisect_left`).
3. If *pos* > 0, all problems P_{prev} with $\text{idx}(P_{\text{prev}}) \in$ `difficulty_seq`$[0 \ldots pos - 1]$ are considered as being on an increasing difficulty path leading to P_{current}. For each such P_{prev}, the counter for the potential prerequisite relationship $(P_{\text{prev}} \to P_{\text{current}})$ in a global $N_P \times N_P$ matrix, `prereq_counts`, is incremented.
4. The `difficulty_seq` is updated: if `pos == len(difficulty_seq)`, $\text{idx}(P_{\text{current}})$ is appended. Otherwise, `difficulty_seq[pos]` is replaced by $\text{idx}(P_{\text{current}})$.
5. A counter, `total_solvers_lis[`$\text{idx}(P_{\text{current}})$`]`, is incremented, tracking how many times P_{current} has been part of these LIS-path constructions.

The overall process is summarized in Algorithm 1.

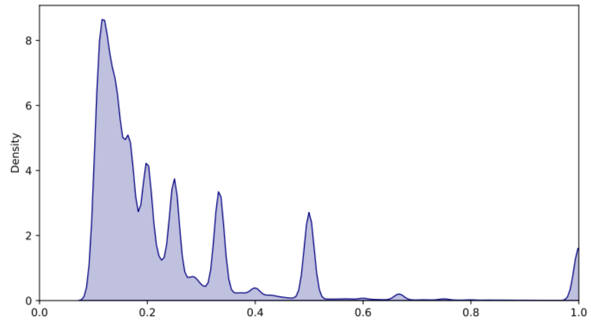

Figure 3: Edge weight distribution in the LIS-derived prerequisite graph. The distribution shows many weak links and fewer strong ones.

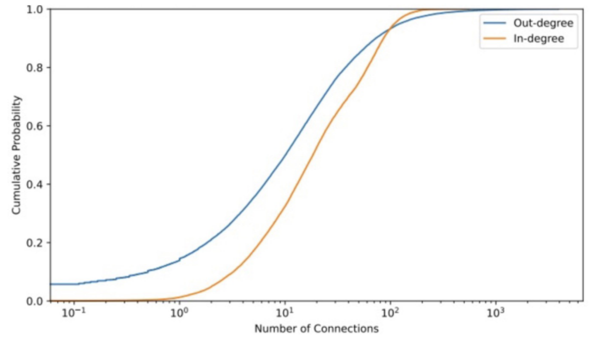

Figure 4: In-degree and out-degree distributions of the LIS-derived prerequisite graph. Most problems have few direct prerequisites or serve as prerequisites for few others, with some hub problems.

Prerequisite Score Normalization and Filtering

After processing all users, the raw `prereq_counts` matrix is normalized. The score $S(P'_i, P'_j)$ for problem P'_i being a prerequisite for P'_j is calculated as:

$$S(P'_i, P'_j) = \frac{\texttt{prereq\_counts}[\mathrm{idx}(P'_i), \mathrm{idx}(P'_j)]}{\texttt{total\_solvers\_lis}[\mathrm{idx}(P'_j)]} \quad (6)$$

This ratio represents the proportion of times P'_i was on an LIS-path leading to P'_j, out of all LIS-path occurrences of P'_j. A final threshold (e.g., 0.1) is applied to this normalized matrix to retain only statistically significant prerequisite relationships. The resulting graph is inherently acyclic due to the difficulty ordering used in the LIS construction. Figure 3 shows a typical distribution of these edge weights, and Figure 4 shows the degree distributions.

3.5 Recommendation Strategy

The primary output of our DKT model at any point in a user's sequence is the "knowledge vector" $\hat{\mathbf{y}}$, representing the predicted probability of solving each problem in the dataset. Our recommendation strategy aims to guide users towards their "knowledge frontier."

A straightforward heuristic based on this vector is to recommend the most challenging problem that the user is predicted to be able to solve with a reasonable probability. Specifically, for a user with current knowledge vector $\hat{\mathbf{y}}$:

1. Filter the set of all problems to include only those not yet successfully solved by the user.
2. From this filtered set, select candidate problems P_j for which the predicted solvability $\hat{\mathbf{y}}[j] > \theta_{solve}$ (where θ_{solve} is a predefined confidence threshold, e.g., 0.6 or 0.7).
3. Among these candidates, recommend the one(s) with the highest official 'problem_rating' (difficulty).

This strategy directly leverages the DKT model's ability to assess current mastery. More advanced strategies, such as recommending problems that maximize Expected Knowledge Gain (EKG), are discussed as future work (Section 6). The explainability of recommendations stems from being able to state the predicted solvability and the potential impact on the knowledge vector.

4 Experiments

To evaluate our proposed Deep Knowledge Tracing (DKT) Transformer model and the impact of incorporating prerequisite graph embeddings, we conducted experiments on a large-scale dataset derived from Codeforces user submission logs. We compare our approach against a traditional Neural Collaborative Filtering (NCF) model adapted for a temporal evaluation setting.

4.1 Experimental Setup

Dataset and Preprocessing. We use the Codeforces interaction dataset described in Section 3.1, containing approximately 17 million submission records from around 15,000 users across over 30,000 unique problems. Each interaction is characterized by the problem ID, the binary outcome (solved/failed), the problem's official rating, the user's Codeforces rating at the time of submission, and the time delta from the user's previous interaction. Continuous features are normalized (Section 3), and problem IDs are label-encoded, resulting in $N_P \approx 30,614$ unique problems.

Train-Test Split. We employ a strict **user-level temporal split**. The dataset is divided such that 80% of users (11,965 users) are randomly selected for the training set, and all their historical interactions are used for training. The remaining 20% of users (2,992 users) form the test set, and their entire histories are used for evaluation. This ensures that the model is evaluated on its ability to generalize to entirely new users it has not encountered during training (a cold-start user scenario). For all users, sequences are processed chronologically. Sequences with fewer than two interactions are filtered out before creating datasets for the models.

DKT Model Configuration. Our primary model is a Deep Knowledge Tracing (DKT) Transformer, whose architecture is detailed in Section 3.2. The model processes sequences of user interactions, with a maximum sequence length informed by our dataset analysis (see Figure 5). Specific hyperparameters for the DKT model, including embedding dimensions, Transformer layer configurations, optimizer details, and training epochs, are provided in Appendix A (Table 2).

DKT Enhancement with Prerequisite Graph Embeddings (DKT+TransE). To investigate the utility of explicit structural knowledge, we evaluate a variant of our DKT model (DKT+TransE). This variant incorporates pre-trained TransE embeddings as additional input features for each problem in a user's interaction sequence. These TransE embeddings are derived from a data-driven prerequisite graph constructed using an LIS-based method on the training data. Further details on the prerequisite graph construction are in Appendix B, and TransE embedding specifics are in Appendix A.

4.2 Baseline Model: Adapted Neural Collaborative Filtering (NCF)

To provide a comparative baseline, we adapt a standard Neural Collaborative Filtering (NCF) architecture [He *et al.*, 2017]. The NCF model is trained on individual '(user_id, problem_id, outcome)' interactions from the same training users as the DKT models. It learns user and problem embeddings and uses an MLP to predict the binary outcome. For evaluation on the cold-start test users, default predictions are used as the NCF has no prior information on these users. Key NCF hyperparameters are detailed in Appendix A (Table 2). This setup ensures the NCF is evaluated under the same challenging cold-start user conditions.

4.3 Baseline Model: Adapted Neural Collaborative Filtering (NCF)

To provide a comparative baseline, we adapt a standard Neural Collaborative Filtering (NCF) architecture [He *et al.*, 2017].

- **Training:** The NCF model is trained on individual '(user_id, problem_id, outcome)' interactions from the same 80% of users in the DKT training set. It learns user and problem embeddings (dimension 64) and uses an MLP to predict a single logit for the binary outcome. User IDs for NCF are separately label-encoded from the 11,965 training users.

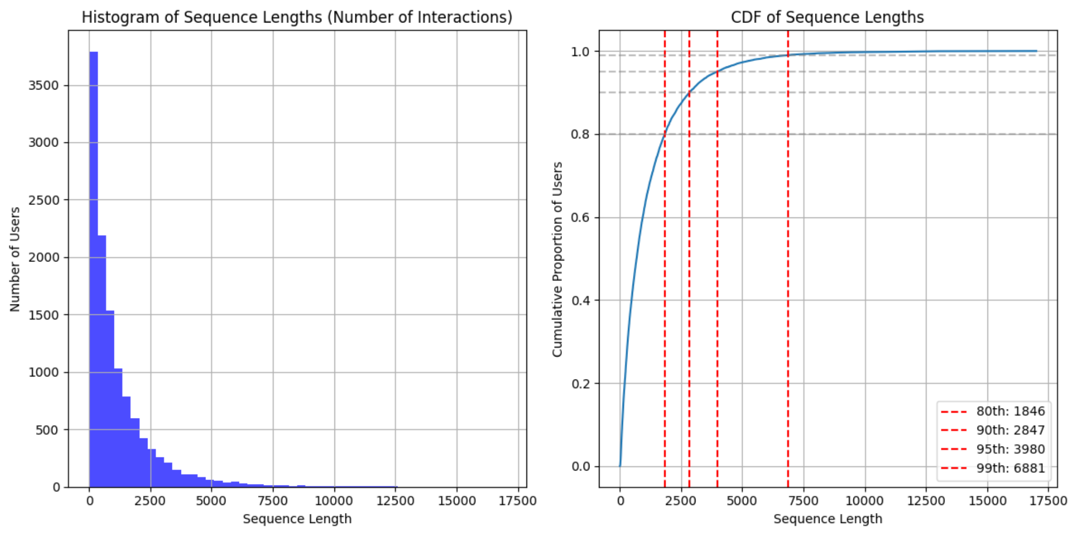

Figure 5: Distribution of user interaction sequence lengths in the training set. Left: Histogram of sequence lengths. (Right) Cumulative Distribution Function (CDF) with 80th (1846), 90th (2847), 95th (3980), and 99th (6881) percentiles marked. The median sequence length is 688. This distribution informed our choice of 'MAX_SEQ_LENGTH' (see Appendix A).

- **Evaluation (Temporal, Cold-Start):** For evaluation, we iterate through the test users' sequences (the 2,992 unseen users). At each step t in a test user's sequence, the NCF model attempts to predict the outcome for the target problem P_{t+1}. Since these are cold-start users for NCF, predictions for ROC AUC, MAE, and RMSE are based on a default zero logit (implying 0.5 probability). HITS@K performance is expected to be minimal. The NCF model was trained for 15 epochs.

This setup ensures the NCF is evaluated under the same challenging cold-start user conditions as the DKT model.

4.4 Evaluation Metrics

We evaluate the models on their ability to predict the outcome of the next interaction in a user's sequence.

- **Area Under the ROC Curve (AUC):** Primary metric for discriminating whether a user will correctly solve the *specific problem they attempt next*.
- **Root Mean Squared Error (RMSE) and Mean AbsoluteError (MAE):** Measure the difference between the predicted probability (after sigmoid) and the actual binary outcome (0 or 1).
- **HITS@K (K=10):**
  - *HITS@10 (Raw Solvability Ranking):* Ranks all N_P candidate problems (not yet solved by the user) based purely on the model's predicted solvability scores (p_j).
  - *HITS@10 (Optimal Challenge Ranking):* Filters candidate problems where DKT's $p_j \in [0.6, 0.9]$, then ranks these by descending official 'problem_rating'. A hit occurs if the user's actual next *solved* problem is in the top K.
- **Loss:** Validation Binary Cross-Entropy loss.

4.5 Results and Analysis

Table 1 presents the performance of our DKT Transformer model, its variant augmented with TransE prerequisite embeddings (DKT+TransE), and the NCF baseline on the cold-start user test set, reporting the metrics from the epoch with the best validation AUC for the DKT models.

DKT Performance. Our DKT-Transformer model demonstrates a strong ability to trace user knowledge and predict solvability for unseen users. As shown in Table 1, the base DKT model achieves a validation AUC of 0.7528 (at epoch 13), significantly outperforming the NCF baseline's AUC of 0.5, which struggles with cold-start users as expected. The DKT model's MAE of 0.3959 and RMSE of 0.2004 indicate reasonable calibration of its probability predictions. This highlights the DKT model's capacity to learn generalizable patterns of skill acquisition from sequential data. The training and validation losses for the DKT model (e.g., DKT-Base validation loss of 0.5853 at epoch 13) consistently decreased over training, indicating stable learning.

Impact of Prerequisite Embeddings (TransE). Incorporating TransE embeddings derived from the LIS-prerequisite graph (DKT+TransE) yielded a slight improvement in performance. The DKT+TransE model achieved a peak validation AUC of 0.7542 (at epoch 14), with a validation loss of 0.5844, MAE of 0.3948, and RMSE of 0.2. While the improvement over the DKT-Base model is modest, it suggests that providing the DKT model with explicit structural knowledge about problem dependencies can offer a small benefit to its learning process and predictive accuracy for new users. The HITS@10 metrics remained very low and similar to the base DKT model.

HITS@K Analysis. The "HITS@K (Raw Solvability Ranking)," which ranks all candidate problems solely by the DKT's predicted $P(\text{solve})$ score, remains very low for both DKT models (around 0.0001). This is expected given the vast problem space ($N_P \approx 30,614$) and the diverse factors influencing a user's next choice beyond raw solvability. Similarly,

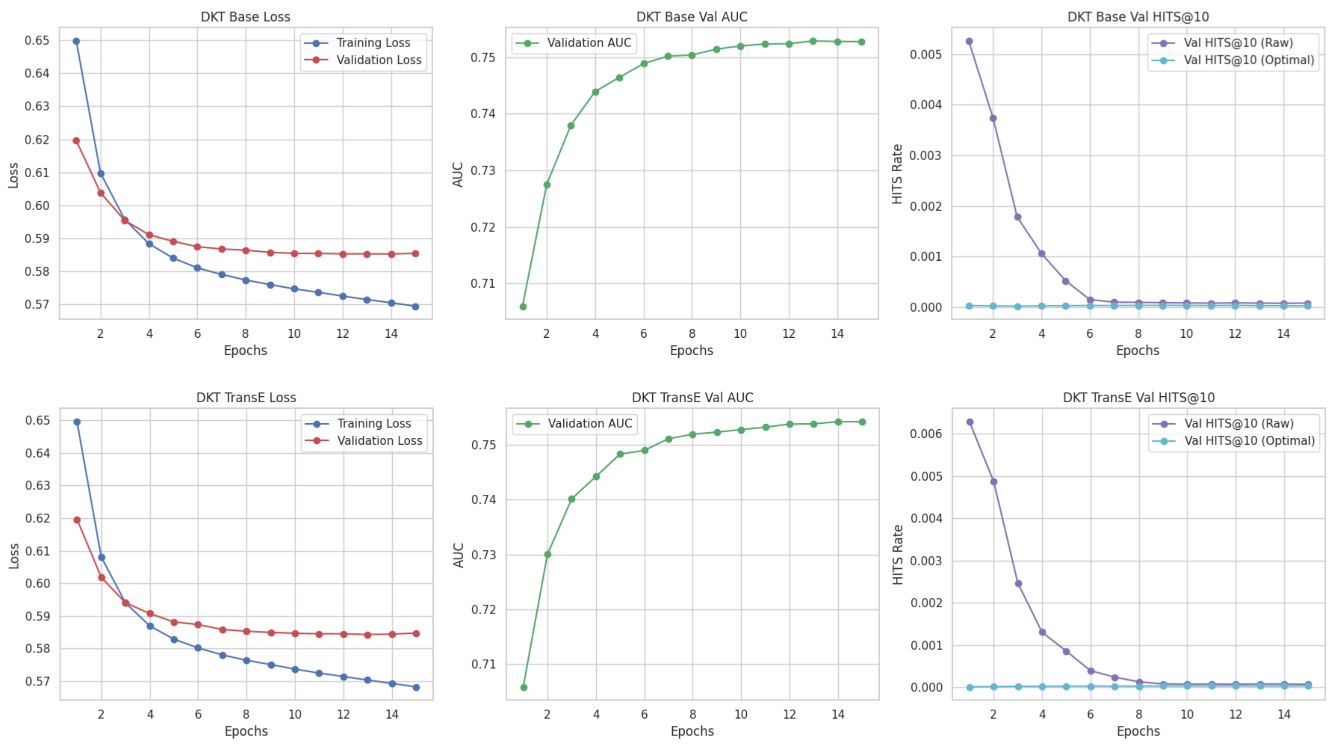

Figure 6: Training and validation performance curves for the DKT-Base and DKT-TransE models over 15 epochs. Shows Training Loss, Validation Loss, and Validation AUC. The model demonstrates stable learning and generalization, with validation AUC reaching 0.7542 for DKT-TransE and 0.7528 for DKT-Base.

Model	Val. Loss	Val. AUC	Val. MAE	Val. RMSE	Val. HITS@10 (Raw)	Val. HITS@10 (Optimal)
NCF (Adapted, Ep. 15)	N/A	0.5000	N/A	N/A	0.0000 (0 / All)	N/A
DKT-Transformer (Base, Ep. 13)	0.5853	0.7528	0.3959	0.2004	0.0001	0.0000
DKT-Transformer + TransE (Ep. 14)	0.5844	0.7542	0.3948	0.2000	0.0001	0.0000

Table 1: Performance comparison on the cold-start user test set. DKT models report best validation AUC epoch. NCF Val Loss/MAE/RMSE are N/A as evaluation focused on AUC/HITS for this baseline. HITS@10 (Optimal) is not applicable to NCF's output.

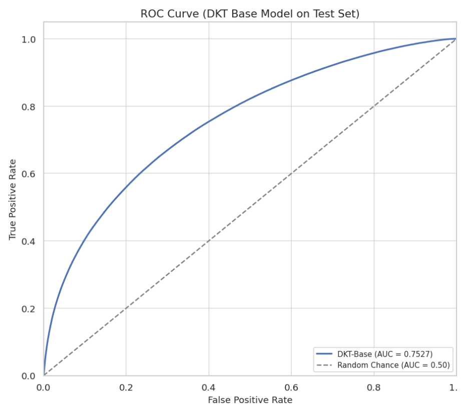

Figure 7: ROC curve for the DKT-Base model on the cold-start user test set. The curve illustrates the model's ability to discriminate between problems the user is likely to solve correctly versus incorrectly. The Area Under the Curve (AUC) achieved is 0.7528.

the "HITS@K (Optimal Challenge Ranking)" also resulted in 0.0000 for both DKT variants in these initial 15 epochs. This suggests that either the model's probabilities are not yet sufficiently calibrated to precisely fit the $[0.6, 0.9]$ "optimal" window, or that users' actual next solved problems do not frequently align with this specific heuristic ranking when considering all possible problems. As discussed previously, for an educational recommender, HITS@K is likely more insightful when evaluating specific recommendation policies built *upon* the DKT's knowledge vector. This remains an important direction for future work (Section 6).

Qualitative Insights Explainability. The DKT model's primary output, the knowledge vector (solve probabilities), provides a strong basis for explainable recommendations. For instance, a problem P_{rec} can be recommended because:

1. The model predicts a high probability of the user solving P_{rec} (e.g., $P(\text{solve } P_{rec}|\text{History}) > \theta_{solve}$).
2. Solving P_{rec} is projected by the DKT model (by simulating the solve and observing the change in the knowledge

vector) to lead to a significant "knowledge gain" on other, potentially harder, problems.

This allows the system to move beyond "users who solved X also solved Y" towards explanations grounded in the user's dynamically tracked skill profile and the potential learning impact of the recommendation.

Qualitative Case Study: Knowledge Evolution. To illustrate how our DKT-Transformer model traces knowledge, Figure 1 depicts the predicted probability of solving a fixed set of initial problems for a sample test user, evolving over their first 50 interactions (x-axis).

Initially, at step 1, the model predicts a low solve probability for an easy problem, say $P1980A$ on the sixth row (an easy 800-level math problem).

Suppose the user then solves this problem successfully, taking two tries for it (indicated by first an X-mark, indicating failure, and then a green circle, indicating success). We see that upon the first failure, the solve probabilities for all other problems remain low. But at the success upon the second attempt, the solve probabilities become much higher.

Following this successful interaction, we might observe in the heatmap that the predicted solve probability for $P1986A$ (an 800-level geometry problem) at step 2 increases from roughly 0.4 to 0.7.

Later, at step 3 the user solves another relevant problem $P1980B$ (a 800-level sorting problem) causing the probability for $P1986A$ to further increase and bringing it into a "solvable" range.

This step-by-step visualization demonstrates the model's ability to dynamically update its assessment of the user's mastery based on their performance, capturing the learning process. It also provides an interpretable basis for why $P1968A$ might be recommended after $P1980A$ and $P1980B$, as the model infers that earlier successes on those problems have increased the user's readiness for $P1986A$.

5 Explainability and Causal Interpretation

A key advantage of our DKT-Transformer approach is its capacity to offer explainable recommendations and facilitate causal-style interpretations of learning pathways. Unlike traditional collaborative filtering models that often function as black boxes, our DKT model's foundation in tracing a dynamic knowledge state provides a more transparent and interpretable basis for its suggestions.

The Knowledge Vector: A Foundation for Explainability. At any step t in a user's interaction sequence, the DKT model outputs a **knowledge vector**, $\hat{\mathbf{y}}_t \in [0,1]^{N_P}$. Each element $\hat{\mathbf{y}}_t[j]$ represents the predicted probability that the user will correctly solve problem P_j, given their history up to step t:

$$\hat{\mathbf{y}}_t[j] = P(\text{user solves problem } P_j | \text{History}_t) \quad (7)$$

This vector provides an immediate, interpretable snapshot of the user's current capabilities across the entire problem set, as estimated by the model.

This knowledge vector directly enables several types of explanations for a recommended problem P_{rec}:

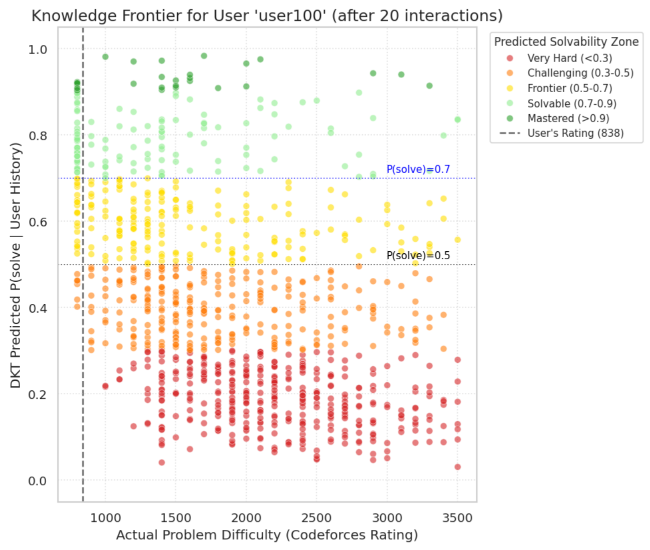

Figure 8: DKT "Knowledge Frontier" for a sample user after k interactions (e.g., $k = 20$). Each point represents a problem, plotted by its actual Codeforces difficulty rating (x-axis) and the DKT model's predicted probability of the user solving it (y-axis). Colors indicate predicted solvability zones: e.g., green for 'Mastered' ($P(solve) > 0.9$), light green for 'Solvable' ($0.7 - 0.9$), gold for 'Frontier' ($0.5 - 0.7$), orange for 'Challenging' ($0.3 - 0.5$), and red for 'Very Hard' (< 0.3). The dashed vertical line indicates the user's Codeforces rating at that time.

- **Solvability-based Explanation:** "P_{rec} is recommended because, based on your interaction history, our system predicts a probability of $\hat{\mathbf{y}}_t[\text{index}(P_{rec})]$ of 0.75 that you can solve it successfully."
- **Knowledge Frontier Explanation:** If using the heuristic from Section 3.5, "P_{rec} is suggested as an appropriately challenging problem predicted to be within your current capabilities ($\hat{\mathbf{y}}_t[\text{index}(P_{rec})] > \theta_{solve}$)."

Simulating Interventions: Estimating Knowledge Gain. The dynamic nature of the DKT model allows us to simulate the impact of hypothetical future actions, a step towards causal reasoning. We can estimate the pedagogical value of attempting a candidate problem P_c by quantifying the "Knowledge Gain" (KG) it might yield. Let $\hat{\mathbf{y}}_t$ be the user's current knowledge vector. If the user were to successfully solve a candidate problem P_c (not yet solved), their history would be augmented: $\text{History}_{t+1} = \text{History}_t \oplus (P_c, \text{outcome} = 1, \text{features}_c)$. Feeding this History_{t+1} into the DKT model yields a new predicted knowledge vector, $\hat{\mathbf{y}}'_{t+1}$. The Knowledge Gain from solving P_c can be defined as the aggregate positive impact on the predicted solvability of other relevant problems (e.g., unsolved problems $\mathcal{P}_{unsolved}$):

$$KG(P_c|\hat{\mathbf{y}}_t) = \sum_{j \in \mathcal{P}_{unsolved}, j \neq c} \max(0, \hat{\mathbf{y}}'_{t+1}[j] - \hat{\mathbf{y}}_t[j]) \quad (8)$$

This measures the total increase in predicted mastery across other problems due to solving P_c.

To account for the user's current ability to solve P_c, we can

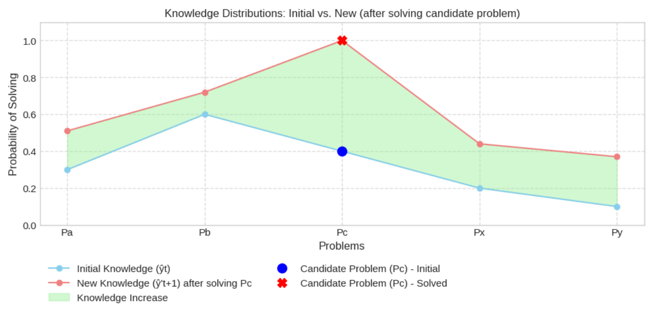

Figure 9: Conceptual illustration of factors contributing to Expected Knowledge Gain (EKG). The 'Initial Knowledge ($\hat{\mathbf{y}}_t$)' curve (light blue) represents the user's baseline estimated probabilities of solving different problems. The candidate problem (P_c) is marked (blue circle) on this initial curve. If P_c is hypothetically solved, its probability becomes 1.0 (marked by a red 'X'). This leads to a 'New Knowledge ($\hat{\mathbf{y}}'_{t+1}$)' curve (red), which generally shows increased probabilities for other problems. The shaded 'Knowledge Increase' area between the two curves visually represents these gains. The sum of the increases in probabilities for problems other than P_c forms the Knowledge Gain, $KG(P_c)$. The Expected Knowledge Gain for P_c is then $EKG(P_c) = \hat{\mathbf{y}}_t[c] \times KG(P_c)$, where $\hat{\mathbf{y}}_t[c]$ is the initial probability of solving P_c.

use the concept of **Expected Knowledge Gain (EKG)**:

$$EKG(P_c|\hat{\mathbf{y}}_t) = \hat{\mathbf{y}}_t[c] \cdot KG(P_c|\hat{\mathbf{y}}_t) \tag{9}$$

This weights the potential knowledge gain by the likelihood of actually achieving that gain (i.e., solving P_c). Problems with high EKG are those that are both achievable and pedagogically impactful.

Recommendations based on EKG can be explained as: "Problem P_{rec} is recommended because it offers a high Expected Knowledge Gain. This means you have a good chance of solving it (predicted probability $\hat{\mathbf{y}}_t[\text{index}(P_{rec})]$), and doing so is predicted to substantially improve your ability to tackle other problems by KG." This provides a powerful, causal-style justification.

Interpreting Broader Knowledge States. The knowledge vector $\hat{\mathbf{y}}_t$, while high-dimensional, can be analyzed to understand a user's proficiency across different problem categories or topics (if such metadata is available and mapped to problems). Visualizing changes in average solvability for these categories over time can illustrate a user's learning trajectory as captured by the DKT model.

Limitations and True Causality. It is important to acknowledge that while our DKT model enables interpretable explanations and causal-style reasoning based on its learned understanding of knowledge evolution, it learns from observational data. The "causal" interpretations are thus based on learned correlations and simulated interventions within the model's learned world. Establishing true causal effects (e.g., that recommending via EKG causes faster learning than another strategy) would necessitate controlled experiments, such as A/B testing different recommendation strategies derived from the DKT model. Nevertheless, the DKT framework provides a significantly richer basis for generating and testing hypotheses about causal learning effects compared to traditional recommendation models.

6 Conclusion

In this paper, we proposed and evaluated a Deep Knowledge Tracing (DKT) system using a Transformer architecture for recommending Codeforces problems. Our primary goal was to move beyond traditional correlation-based methods by modeling the temporal evolution of a user's knowledge state to provide more pedagogically sound and explainable recommendations. Crucially, our approach outputs a dynamic "knowledge vector" that both predicts user solvability and quantifies expected knowledge gain, enabling transparent, pedagogically informed suggestions.

Using a large-scale dataset of Codeforces interactions, we demonstrated that our DKT-Transformer model can effectively trace user knowledge, achieving a validation AUC of 0.7542 in predicting the outcome of the next problem a user attempts, significantly outperforming a standard NCF baseline in a cold-start user scenario. We also explored the integration of explicit prerequisite knowledge by incorporating TransE embeddings derived from a data-driven prerequisite graph, with DKT+TransE showing a small but measurable improvement over just DKT alone.While HITS@K remains a stringent test of exact-match recommendations, it provides useful insight when evaluating intervention policies grounded in our tracing framework.

Our work underscores the potential of sophisticated sequential models like Transformers for DKT in complex domains like competitive programming. By focusing on an accurate representation of the learner's evolving capabilities, we can build recommender systems that not only suggest relevant content but also provide transparent reasons for their suggestions, fostering a more effective and understandable learning experience.

Future Work. The DKT-based framework presented opens several exciting avenues for future research. A primary direction is the development and rigorous evaluation of recommendation policies based on maximizing Expected Knowledge Gain (EKG), where problems are ranked by $P(\text{solve}) \cdot$ KnowledgeGainIfSolved; comparing such policies using appropriate HITS@K metrics will be crucial for optimizing learning. We also plan to explore using the trained DKT model's learned dynamics to automatically infer or refine a more nuanced problem prerequisite graph, for instance, by analyzing asymmetrical influence patterns where solving one problem boosts solvability of another. For users with specific long-term goals, investigating A* search or similar pathfinding algorithms operating on a problem graph, guided by the DKT model's knowledge state and EKG-related heuristics, could generate optimal, personalized learning paths. Finally, experimenting with more recent Transformer variants designed for longer sequences or improved relational reasoning may further enhance the DKT model's knowledge tracing capabilities. Pursuing these directions aims to significantly advance intelligent tutoring and recommendation systems in complex problem-solving domains.

7 Acknowledgments

This work was partially supported by NSF 2211557, NSF 2119643, NSF 2303037, NSF 2312501, SRC JUMP 2.0 Center, Amazon Research Awards, and Snapchat Gifts.

References

[Bordes *et al.*, 2013] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*, pages 2787–2795, 2013.

[Chen *et al.*, 2018] Shuanghong Chen, Qi Liu, Zhenya Huang, Enhong Wu, Wei Wang, Yu Su, and Guoping Chen. Prerequisite-driven deep knowledge tracing. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 857–862, 2018.

[Coc, 2021] Codeforces coco: Problem suggestion. https://devpost.com/software/codeforces-problem-suggestion, 2021. Accessed: 2025-04-01.

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

[da Silva *et al.*, 2023] Felipe Leite da Silva, Bruna Kin Slodkowski, Ketia Kellen Araújo da Silva, and Sílvio César Cazella. A systematic literature review on educational recommender systems for teaching and learning: research trends, limitations and opportunities. *Education and information technologies*, 28(3):3289–3328, 2023.

[Denk, 2024] Denk. Users codeforces submissions end 2024 dataset. https://huggingface.co/datasets/denkCF/UsersCodeforcesSubmissionsEnd2024, 2024. Accessed: 2025-04-01.

[Desai, 2023] Brij Desai. Codeforces problem recommender. https://github.com/brij-desaii/Codeforces-Problem-Recommender, 2023. Accessed: 2025-04-01.

[Goldberg *et al.*, 1992] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[He *et al.*, 2017] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*, pages 173–182, 2017.

[Hidasi *et al.*, 2016] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

[Kang and McAuley, 2018] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206, 2018.

[Kirschner *et al.*, 2006] Paul A Kirschner, John Sweller, and Richard E Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist*, 41(2):75–86, 2006.

[Kumar *et al.*, 2023] Nischal Ashok Kumar, Wanyong Feng, Jaewook Lee, Hunter McNichols, Aritra Ghosh, and Andrew Lan. A conceptual model for end-to-end causal discovery in knowledge tracing. *arXiv preprint arXiv:2305.16165*, 2023.

[Li *et al.*, 2022] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Robson Sutherland, Oriol Vinyals, R. Hadsell, C. Apps, A. Muldal, R. Ring, and T. Graepel. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[Pandey and Karypis,] Shalini Pandey and George Karypis. A self-attentive model for knowledge tracing. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining (WSDM)*, pages 625–633.

[Piech *et al.*, 2015] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J. Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*, pages 505–513, 2015.

[Rendle *et al.*, 2010] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 811–820, 2010.

[Resnick *et al.*, 1994] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews, 1994.

[Roy *et al.*, 2019] Sudeshna Roy, Meghana Madhyastha, Sheril Lawrence, and Vaibhav Rajan. Inferring concept prerequisite relations from online educational resources. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 9589–9594, 2019.

[Sarwar *et al.*, 2001] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.

[Schnabel *et al.*, 2016] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten

Joachims. Recommendations as treatments: Debiasing learning and evaluation. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1670–1679, 2016.

[Shin *et al.*, 2021] Dongmin Shin, Chanyou Lee, JeongA Heo, Hyeong-Sic Chae, Yunki Kim, and Youngduck Choi. SAINT+: A self-attentive interaction tagger for knowledge tracing. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, pages 3513–3517, 2021.

[Sun *et al.*, 2019] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1441–1450, 2019.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 5998–6008, 2017.

[Wang *et al.*, 2014] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*, pages 1112–1119, 2014.

[Wang *et al.*, 2019] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 950–958, 2019.

[Wang *et al.*, 2023] Yongfeng Wang, Xu Zhang, and Qing Wang. Causal inference in recommender systems: A survey. *ACM Transactions on Recommender Systems*, 1(2), 2023.

[Wu *et al.*, 2022] Shiwen Wu, Fuli Feng, Xiangnan He, Tianxin Wei, Chonggang Song, Guohui Ling, and Yongdong Zhang. Graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Transactions on Recommender Systems*, 1(1), 2022.

[Ying *et al.*, 2018] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 974–983, 2018.

[Zhang and Chen, 2020] Yongfeng Zhang and Xu Chen. Explainable recommendation: A survey and new perspectives. *Foundations and Trends® in Information Retrieval*, 14(1):1–101, 2020.

[Zhang *et al.*, 2021] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and Fuli Feng. Causal inference in recommender systems: A tutorial. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, pages 4755–4758, 2021.

A Hyperparameters

Parameter Group	Setting / Value
DKT-Transformer (Base & +TransE unless specified)	
Embedding Dimension (d_{model})	128
Attention Heads (N_{head})	4
Transformer Encoder Layers	2
Feed-Forward Dimension (Encoder)	256
Dropout Rate	0.1
Max Sequence Length (`MAX_SEQ_LENGTH`)	512
Optimizer	AdamW
Learning Rate	1×10^{-4}
Loss Function	Binary Cross-Entropy
Training Epochs (Max)	15
Checkpointing	Based on best validation AUC
Prerequisite Graph Embeddings (for DKT+TransE variant)	
Source Graph Construction	LIS-based on training data (see Appendix B)
TransE Embedding Dimension (d_{emb})	64
NCF Baseline Model	
User Embedding Dimension	64
Problem Embedding Dimension	64
MLP Hidden Dimensions	[128, 64]
NCF Dropout Rate	0.3
Optimizer	Adam
Learning Rate	1×10^{-3}
Loss Function	Binary Cross-Entropy
Training Epochs	15

Table 2: Key hyperparameters and configuration details for the DKT models and the NCF baseline. Specifics of prerequisite graph construction for TransE are in Appendix B.

Algorithm 1 Prerequisite Graph Construction

```
0: Initialize empty prerequisite matrix $S \in R^{P \times P}$
0: Compute problem difficulty ordering Order$(p)$ for all problems
0: Create problem-to-index mapping idx$(p)$
0: for each user $u \in$ Users do
0:     Initialize empty sequence $L = []$
0:     for each problem $p$ solved by $u$ in chronological order do
0:         if $p$ not in idx or already processed then
0:             continue
0:         end if
0:         $i \leftarrow$ idx$(p)$
0:         $k \leftarrow$ bisect_left$(L, i)$
0:         if $k >$ then
0:             $S[L[1..k], i]$ += 1
0:         end if
0:         Update $L$ with $i$ at position $k$
0:         total_solvers$[i]$ += 1
0:     end for
0: end for
0: Normalize $S$ by total_solvers
0: Apply threshold $\tau$ to $S$
0: return $S$ =0
```

B Prerequisite Graph Construction and Embedding

This appendix details the methodology used to construct the problem prerequisite graph from user interaction data and the subsequent generation of TransE embeddings for these problems. This graph and its embeddings can be used as features to enhance the Deep Knowledge Tracing (DKT) model by providing explicit structural information about problem dependencies.

B.1 TransE for Prerequisite Embeddings

To encode the structural information from the generated prerequisite graph into vector representations suitable for machine learning models, we employ TransE [Bordes *et al.*, 2013], a knowledge graph embedding technique.

Triple Formulation

The prerequisite graph, where an edge $A \to B$ signifies that A is a prerequisite for B, is transformed into a set of (head, relation, tail) triples. In our context, we primarily model a single relationship type, "isPrerequisiteOf". Thus, each significant edge $A \to B$ (after thresholding the LIS-derived scores) becomes a triple $(A, \text{isPrerequisiteOf}, B)$. Problem IDs are mapped to unique integer entity IDs for the embedding model.

TransE Objective and Training

TransE learns d_{emb}-dimensional vector embeddings for entities (problems, $\mathbf{e}_P$) and relations ($\mathbf{r}$). The fundamental principle is that if a relation r holds between a head entity h and a tail entity t, their embeddings should satisfy $\mathbf{e}_h + \mathbf{r} \approx \mathbf{e}_t$. The model is trained by minimizing a margin-based ranking loss. This loss function aims to ensure that the score for valid triples (positive samples) is lower than the score for corrupted triples (negative samples), where negative samples are generated by replacing either the head or tail of a valid triple with a random entity. A common scoring function is the L1 or L2 norm of $(\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t)$. We initialize entity and relation embeddings (e.g., using Xavier initialization) and train the TransE model using an optimizer like Adam. For our single "isPrerequisiteOf" relation, its embedding $\mathbf{r}$ captures the typical vector translation associated with this dependency.

Output Embeddings

After training, the TransE model yields an embedding matrix where each row corresponds to the learned vector $\mathbf{v}_P^{TransE} \in R^{d_{emb}}$ for a problem P. These embeddings capture the relational structure of the prerequisite graph and are used as input features to our DKT model, as detailed in Section 3.3, to provide it with explicit structural priors about problem dependencies.