Error Tracing in Programming: A Path to Personalised Feedback

Martha Shaka, Diego Carraro and **Kenneth N. Brown**
Centre for Research Training in AI, Insight, School of Computer Science & IT,
University College Cork
Ireland
m.shaka@cs.ucc.ie, diego.carraro@insight-centre.org, k.brown@cs.ucc.ie

Abstract

Knowledge tracing, the process of estimating students' mastery over concepts from their past performance and predicting future outcomes, often relies on binary pass/fail predictions. This hinders the provision of specific feedback by failing to diagnose precise errors. We present an error-tracing model for learning programming that advances traditional knowledge tracing by employing multi-label classification to forecast exact errors students may generate. Through experiments on a real student dataset, we validate our approach and compare it to two baseline knowledge-tracing methods. We demonstrate an improved ability to predict specific errors, for first attempts and for subsequent attempts at individual problems.

1 Introduction

The increasing importance of digital technologies has made programming a critical skill. The teaching of programming has long been recognised as difficult, and novice programmers often struggle with syntax, and with conceptual and problem-solving skills (Figueiredo and García-Peñalvo, 2021; Thuné and Eckerdal, 2019). Practical assignments, designed to enhance understanding, often become stumbling blocks due to compiler errors that are not informative for beginners, leading to confusion or discouragement (Medeiros et al., 2019). Further, given large class sizes, providing personalised feedback from instructors is difficult (Parihar et al., 2017; Song et al., 2019). Recent research has explored Automatic Feedback generation, including test-case analysis (Xiong et al., 2018) and AI-driven Automatic Program Repair systems (Bhatia and Singh, 2016; Gulwani et al., 2018; Suzuki et al., 2017). But many of these system fail to trace the individual learner's profile or unique learning trajectory, thus reducing the effectiveness of the feedback provided (Ghosh et al., 2021).

In contrast, Knowledge Tracing (KT), an educational data mining technique, has the potential to create personalised learning experiences by predicting student performance based on their mastery of concepts (Piech et al., 2015; Wang et al., 2017; Emerson et al., 2019). In programming education, KT is useful for recommending exercises, predicting assignment outcomes, and identifying students at risk of underperforming (Huang et al., 2019; Azcona et al., 2019). But traditional KT models often overlook the granularity of student responses, treating all correct or incorrect attempts uniformly (Ghosh et al., 2021). Programming errors, though, vary widely, from simple syntax mistakes like a missing semicolon, to more complex issues such as failing to implement a loop correctly. Deep Knowledge Tracing (DKT) (Piech et al., 2015) uses neural networks to identify specific patterns, and thus allows more specific feedback.

This paper propose a refined application of DKT to identify precise compiler errors. By analysing the error patterns in students' historical performance, we aim to identify the specific concepts or syntax elements that a student has not yet mastered. This then enables the delivery of targeted feedback focused on those elements. In addition, by analysing the patterns of multiple students in a class, we can highlight common error patterns, for further action by educators.

Our contributions are as follows. (1) We introduce a novel KT task, error-based knowledge tracing, to learn a meaningful representation of student submissions. We introduce a new error-based deep knowledge tracing model (Error-DKT) to track the progressive student error patterns. (2) We conducted experiments on a real-world student code database and found that incorporating error features significantly enhances the accuracy of specific error predictions, elevating the F1 score from 0.27 (as seen in existing models) to 0.5. (3) We discuss the broader implications and limitations of this research within programming education, proposing new research directions to bridge the gap between generic feedback systems and the need for individualised educational support.

2 Related Work

Knowledge Tracing (KT) is designed to predict students' future performance by analysing their past interactions with learning materials. Initially, KT relied on probabilistic models such as Bayesian Knowledge Tracing (BKT) (Corbett and Anderson, 1994), which estimates students' mastery using a Bayesian Network and a set of fixed parameters (guess, slip, learn, and sometimes forget). BKT's extended by Käser et al. (2017) through the introduction of Dynamic BKT, to account for interactions between different knowledge components.

Deep Knowledge Tracing (DKT) leverages recurrent neural networks to harness the sequential patterns in student interaction data, effectively capturing not only correctness of responses but also the order and context of these interactions (Piech et al., 2015). Recent advances includes techniques such as attention mechanisms (e.g., AKT-Context-aware attentive knowledge tracing (Ghosh et al., 2020)), external memory modules (e.g., DKVMN-Dynamic key-value memory networks for knowledge tracing (Zhang et al., 2017)), and GKT-Graph-based KT (Nakagawa et al., 2019), each aiming to better understand the learning process's complexities. DKT has outperformed recent deep learning models (Shi et al., 2022; Liu et al., 2022). Liu et al. (2023); Abdelrahman et al. (2023) gives a comprehensive review of KT models.

Traditional DKT models primarily rely on sequences of question numbers and the correctness of attempts for prediction, often overlooking detailed information about students' approaches to solving questions (Shi et al., 2022; Ghosh et al., 2021; Abdelrahman et al., 2023). This omission restricts their predictive power across different domains. However, incorporating domain-specific features has been shown to enhance performance. For example, in the mathematical domain, Liu et al. (2020) enhanced predictions by including question-concept relationships derived from Pre-training Embeddings via Bipartite Graph (PEBG), while in the programming domain, Shi et al. (2022) introduced code features using code2vec.

There has been a push to extend DKT's application beyond mere correctness prediction. Ghosh et al. (2021) adapts DKT to forecast the specific options students select in multiple-choice questions. Inspired by this, our work aims to tackle the more complex scenario of open-ended programming questions, which creates the challenge of interpreting diverse compiler errors. Liu et al. (2022) develops Open-ended Knowledge Tracing (OKT), which integrates an enhanced DKT model with code features from an abstract syntax tree neural network-ASTNN (Zhang et al., 2019) and textual question features from GPT-2, aiming to predict student performance. They then employ a GPT-2-based text-to-code generator, guided by the DKT model's hidden state as a knowledge estimate, to generate diverse code solutions that mirror the student's comprehension.

3 Methodology

Our work introduces an alternative approach for DKT to predict directly the specific errors students are likely to encounter. We assess how different domain features like student code submissions, reference solutions, and question-concept relationships affects error prediction. After pinpointing individual errors, we employ a bottom-up approach, aggregating these error predictions to assess overall student performance as pass (error-free) or fail (submission with errors). This is to assess whether focusing on granular error predictions can enhance the accuracy of student outcome forecasts compared to traditional DKT predictions. Figure 1 illustrates our proposed model structure.

3.1 Dataset

We use a dataset from a US university's Spring Semester introductory Java programming course, conforming to ProgSnap2 format (Price et al., 2020). This dataset includes data from 410 students across five assignments, totaling 50 programming questions that assess various concepts like loops and conditions. Students submitted multiple attempts per exercises until achieving a 100% score, with submissions ranging from 10 to 20 lines of code and automatically graded based on test cases.

We focused on Assignment 1, which consists of 10 questions, selected for its high error frequency and variety, providing a comprehensive base for error analysis (details in Table 1). Our analysis employs two subsets: **Set-I**, categorising submissions with compiler errors as *"incorrect"* and those with-

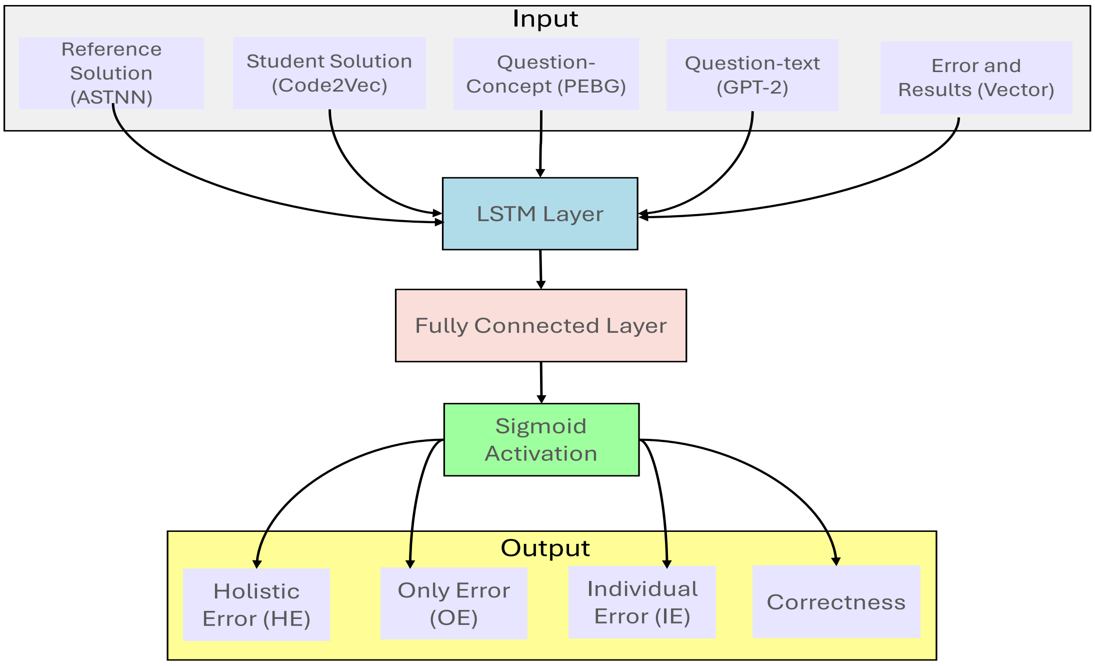

Figure 1: Architecture of Error Tracing, integrating various feature embeddings and LSTM layers to predict student errors and overall performance, as described in section 3.3.

Description	Dataset
Total submissions (subs)	9995
Subs with errors	5948 (59.5%)
Avg errors per subs	1.6
Top 3 frequent errors	[0, 1, 5]
Top 2 common pairs	[1, 3] [1, 2]
Total No of students	386
Avg students per question	368
Most attempted question	5 ($\approx$ 4000 errors)
Least attempted question	4 ($\approx$ 750 errors)

Table 1: Key Features of the Dataset: Summarises submission counts, error rates, common errors, and student engagement metrics, highlighting critical areas of focus within student interactions.

ID	Description	Frequency
0	Passed/ No error	4047
1	'ID' expected e.g like ";)("	2128
2	Missing return statement	1291
3	Illegal start of expression	1163
4	not a statement	850
5	'else' without 'if'	629
6	Cannot find symbol: variable ID	624
7	Bad operand types for binary operator 'ID', like "&&, ‖,*,+,>=,<"	554
8	Incompatible types, like datatypes mismatch	444
9	Reached end of file while parsing, maybe a missing delimiter or closing brace	426

Table 2: Overview of key error types in student submissions, presenting both the frequency and characteristics highlighting common obstacles in the learning process.

out as *"correct"*, specifically for error prediction. **Set-II** is for binary (pass/fail) prediciton, labelling any submission without a perfect score as *"incorrect"* due to compiler or logical errors, and those with full marks as *"correct"*.

To mitigate class imbalance in Set-I, we identified the top 10 errors for proof of concept which includes nine error types and a pass class, with occurrences from 5000 to 400 across the questions, detailed in Table 2.

3.2 Problem Definition

Our approach treats students' code submissions as a temporal sequence, aiming to trace their concept mastery over time. Each submission at time step t is represented as $x_t = \langle p_t, c_t, s_t, e_t, r_t, \{ref\}_t \rangle$, encapsulating the problem p_t, concept c_t, code solution s_t, errors e_t, result (pass/fail) r_t, and reference

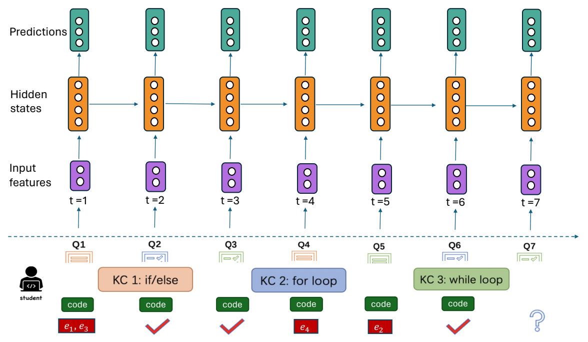

Figure 2: Overview of a simplified RNN Model. The model performs predictions at each timestep, using the previous hidden state (representing estimated mastery) complemented by a diverse input features.

solution $\{ref\}_t$. Given T is the maximum number of attempts, we define students submission as S_T = $\{x_1, x_2, x_3, \ldots, x_T\}$. Our aim is to predict the specific errors e_{T+1} that might arise in the next problem p_{T+1}, based on the student's previous submissions. For example, as illustrated in Figure 2, our target is identifying potential errors (e_7) at time step t_7 based on submissions from t_1 to t_6, while correctness prediction determines the likelihood of a pass/fail result (r_7).

3.3 Error Tracing with DKT

We built an error tracing model (Error-DKT [1]) that utilises a Long Short-Term Memory (LSTM) neural network and a combination of prediction strategies to solve the challenges in multi-label error prediction. As illustrated in Figure 1, it includes constructing detailed input features and a layered architecture, featuring an LSTM layer to discern hidden knowledge states, and a fully connected layer that converts LSTM outputs for multi-label prediction. We investigated two predictive strategies:

Standalone prediction: This strategy employs methods where individual models operate independently to make error predictions.

Holistic Error prediction (HE) a single model is trained to identify probabilities for specific error classes, including a unique "no error" class. This model employs a dynamically adjustable threshold to determine the overall presence/absence status. For instance, if there are four possible error types, the HE model will predict among five outcomes, where one represents the absence of errors

Only Error prediction (OE) focuses solely on detecting errors in a submission. Referring to the example above, OE model will predict the presence of four error classes; if all predictions fall below a certain threshold, the submission is classified as error-free.

Individual Error prediction (IE) a separate model is trained for each error type. Using the same example with four error types, four distinct models would be trained. Their predictions are then aggregated to formulate a comprehensive view of the errors in a student's attempt.

Ensemble Methodology: This two-step approach initially evaluates the likelihood of any error occurrence before pinpointing exact errors using insights gained from the initial assessment. The Ensembled Error Prediction strategy combines the strengths of conventional DKT in determining submission correctness with the detailed error tracing capabilities of our model to isolate precise errors.

3.4 Baseline Models

To tackle the novel challenge of predicting specific programming errors without established benchmarks, we develop two baseline models using statistical probabilities. The **Simple Baseline Model** use overall dataset statistics to forecast error probabilities, identifying the two most frequent errors per question from historical data. In contrast, the **Complex Baseline Model** offers a granular analysis, calculating error probabilities for each question-attempt pair and pinpointing the two most common errors based on historical data, though it overlooks individual error histories. Additionally, we benchmark against the Open-ended Knowledge Tracing **OKT** approach (Liu et al., 2022), which employs a large language model to generate student code. We run that code through a compiler to identify expected errors, excluding the errors not included in our set thus providing a direct comparison with our error tracing model.

4 Experimental Setup

Our experimental setup, detailed below, outlines the data collection methods, model training protocols, and evaluation metrics used to rigorously test the efficacy of our proposed models.

4.1 Data Preprocessing

We grouped the submissions by student and divided them into training and test sets with a ratio of 4:1. A random split method is used for performance prediction, with an iterative stratification

[1]https://github.com/kencyshaka/Error-DKT

technique, specifically *MultilabelStratifiedShuffleSplit* (Sechidis et al., 2011), used to address class label imbalances for error prediction. We further split the training set to allocate 25% for validation, facilitating hyperparameter tuning. The entire training dataset, including the validation subset, was subsequently utilised for model training, with performance evaluation conducted on the test set.

4.1.1 Constructing Input Features

The input feature x_t for each timestep is:

$$\begin{aligned} x_t =& [\mathrm{E_r}(r_t) \oplus (\mathrm{E_p}(p_t) \odot \square) \oplus (\mathrm{E_c}(c_t) \odot \square) \\ & \oplus (\mathrm{E_{ref}}\{ref\}_t \odot \square) \oplus (\mathrm{E_{er}}(\{er\}_t) \odot \square)] \end{aligned} \tag{1}$$

$\odot$ and $\square$ signify element-wise multiplication and the binary presence or absence of embeddings, respectively. $\oplus$ concatenates to create the final embedding, integrating the problem content ($\mathrm{E_p}$), student and reference code ($\mathrm{E_c}$ and $\mathrm{E_{ref}}$), and errors ($\mathrm{E_{er}}$), alongside results ($\mathrm{E_r}$) to effectively predict student performance.

Problem and Code Embeddings Problem representation (E_p) merges textual content (E_{p1}) and concept relationships (E_{p2}) into a comprehensive embedding. E_{p1} leverages a GPT-2 model trained on Java datasets for textual transformation (Liu et al., 2022), while E_{p2} employs a bipartite graph to capture problem-concept dynamics, following the PEBG methodology (Liu et al., 2020).

Code representation adopts ASTNN (Zhang et al., 2019) for the reference solution (E_{ref}) and a modified code2vec (Alon et al., 2019) approach for student submissions (E_c), facilitating dynamic adaptation during model training (Shi et al., 2022).

Categorical Embeddings Categorical features, such as error lists and outcome indicators, are transformed into vector representations. Error lists are encoded into binary vectors (E_{er}), with the vector size reflecting the total number of distinct errors. Similarly, result embeddings (E_r) denote attempt results and question interactions (Piech et al., 2015), utilising a binary format to represent the data efficiently.

4.2 Network Architectures and Hyperparameter Optimisation

We systematically explored hyperparameters to identify the optimal model configuration, assessing their impact on model performance through average loss and F1 scores on the validation dataset. This iterative process, conducted 100 times, aimed to pinpoint the hyperparameter set yielding the best validation results, which was then applied across the entire training set to construct the final model for subsequent testing and evaluation phases.

Input features, including code embeddings ($\mathrm{E_c}$), reference solution embeddings ($\mathrm{E}_{\{ref\}}$), and textual problem embeddings ($\mathrm{E_{p1}}$), were configured following default parameters from prior studies in Code-DKT (Shi et al., 2022) and OKT (Liu et al., 2022). For the problem-concept relationship component ($\mathrm{E_{p2}}$), we use the PEBG framework, varying parameters such as embedding size ($d = \{\mathbf{64}, 128\}$), epochs ($10, 50, \mathbf{100}, 200$), learning rate ($\mathbf{0.001}, 0.005, 0.0015$), hidden states ($\mathbf{128}, 256$), and batch size ($\mathbf{16}, 32, 128$), with the optimal settings highlighted in bold.

Our architecture exploration was tailored to specific tasks, employing varying hyperparameters to refine the model's structure. This included adjustments to LSTM layers ($1, 2, 4, 8, 10$), learning rates (uniform distribution, min=0.00001, max=0.001), batch sizes (16, 32, 64, 128), epochs (10, 20, 40, 50, 70, 100), threshold settings, and loss types (Binary Cross Entropy, Focal Loss (Lin et al., 2018), Class Balanced (Cui et al., 2019) and Distributed Balanced Loss (Wu et al., 2020)). The selected hyperparameters for each multi-label task in Section 3.3 are summarised in Appendix A.3 Table 6.

Model training and evaluation on an NVIDIA A-40 GPU averaged 10 minutes, while the same tasks took about 4 hours on a local CPU. For further details, see Appendix A.3, Table 6. We use the Adam optimizer for learning rate scheduling in training. Consistent with prior research (Shi et al., 2022), we limited the number of student attempts to 50 for each problem, focusing on the most recent submissions to better reflect current understanding and skills.

4.3 Evaluation Metrics

Model performance: The primary metric for error prediction is the weighted average F1 score, tailored to reflect the proportion of each error class within the dataset. This approach guarantees a balanced evaluation, highlighting the model's precision for common errors while proportionally considering less frequent ones. Weighted average precision and recall further detail the model's predictive accuracy. Additionally, we use the weighted average F-beta score, emphasising precision more than recall. This prioritisation is crucial, as it ensures that any predicted errors intended to guide interventions are reliably identified, maximising the relevance and efficacy of educational support. For performance prediction on the correctness (pass/fail), we use the Area Under the Receiver Operating Characteristic curve (AUC) alongside the average F1 score to assess model performance.

Educational Context: We analyse the model's performance in two educational scenarios: overall accuracy and accuracy in predicting the first attempt at solving a problem. The latter is crucial for identifying early intervention opportunities in knowledge tracing (Emerson et al., 2019), while the overall performance metric helps differentiate between types of errors (conceptual vs. syntactical) and debugging skills.

Problem and Error Analysis: Further, we evaluate the model's effectiveness across individual questions to capture how well historical performance data informs future error predictions. We also evaluate the model performance on the most common to the least frequent errors. This analysis is crucial for understanding the model's capacity to predict common errors (easy task) and uncommon errors (hard task).

5 Results

Results are shown in Table 3, for the baselines, the error prediction tasks and the ensemble approaches.

5.1 Error Prediction

Predictive Performance The Error-DKT models, employing single-step and ensemble strategies, outperform baselines, e.g, OKT by +15.8% and +23.2% respectively, showcasing their superior performance in predicting overall student errors. This efficacy is particularly highlighted in the ensemble approach, which underscores the benefit of first identifying error-free submissions before employing Error-DKT models to pinpoint specific student errors, thereby improving overall prediction accuracy. Specifically, focused error prediction models (OE and IE) benefit from this approach, e.g, OE using Distributed Balance loss has +35% increase in accuracy for first attempts.

Performance at predicting first attempt is generally higher than for all attempts, including for OKT (Liu et al., 2022). We believe this is because each question is initially the same for each student, and cohort data for previous questions is informative. Once a student has submitted an attempt that fails, the student is then responding to the compiler messages, and so the task becomes individualised, negating the benefit of more data on each individual student.

Model	**First**		**Overall**	
	F1	F-beta	F1	F-beta
Simple	30.5	32.6	22.9	21.1
Complex	40.9	39.7	26.1	**25.1**
OKT	**47.1**	**41.8**	**27.5**	22.9
HE-BCE	49.2	48.8	42.8	**42.5**
HE-FL	**50.8**	**48.2**	**43.3**	41.8
OE-BCE	20.2	19.8	16.7	17.3
OE-DB	16.5	19.3	30.7	30.3
IE	17.0	19.4	34.1	34.1
HE-BCE	52.0	51.9	43.7	44.8
HE-FL	**53.1**	**53.1**	**50.7**	**50.1**
OE-BCE	52.2	51.2	36.4	37.2
OE-BCE*	52.6	51.6	44.7	44.4
OE-DB	51.5	51.4	45.2	45.7
IE	51.4	51.6	46.9	47.8

Table 3: Evaluation of Model Performance Across Error Prediction Tasks: The table presents F1 and F-beta scores for 'First' and 'Overall' attempts. It starts with baseline model metrics, progresses through Error-DKT error prediction tasks (Holistic Error), OE (Only Error), IE (Individual Error), and concludes with ensemble approaches combining Error-DKT predictions with DKT outcomes. OE-BCE* denotes the model trained solely on submissions with errors. The losses are BCE-Binary Cross Entropy, FL-Focal and DB-Distributed Balance. Bold values highlight top performance within each section.

Interestingly, the holistic approach (HE) demonstrated superior performance over focused error predictions (IE, OE), especially in contexts with limited data and high imbalance, indicating the challenges of granular error prediction. The enhanced performance of IE and OE in overall attempts, as opposed to first attempts, suggests that accumulating more data leads to improved accuracy. Furthermore, utilising various loss functions to tackle class imbalances significantly enhances model performance. For instance, employing Focal loss results in a +1.6 improvement for HE prediction compared to Binary Cross Entropy, and dynamically adjusting thresholds for error classes also contributes to this advancement.

Per Problem The analysis shows variations in predictive model performance, which could be due to the distinct challenges, skill requirements and

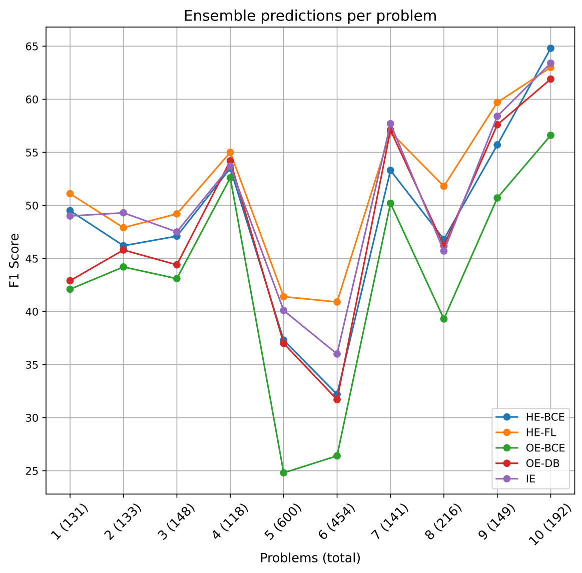

Figure 3: Ensemble Model's Performance per Problem: Overall Attempts.

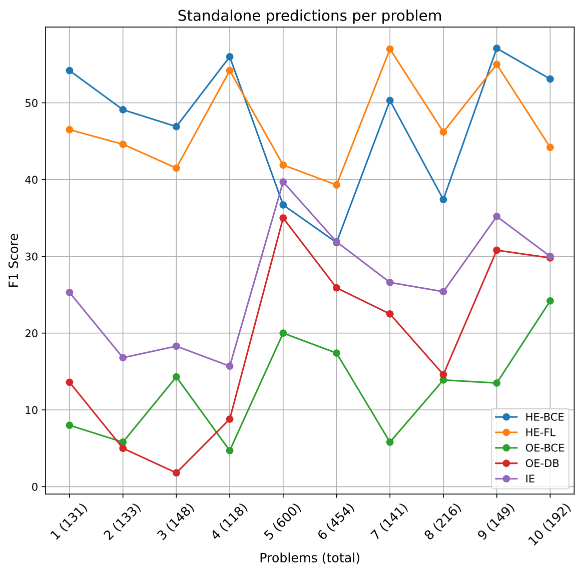

Figure 4: Error-DKT Model's Performance with various Prediction tasks per Problem: Overall Attempts.

prevalence of errors in each problem. A general pattern shows that the model's performance increases in predicting student errors as they advance in their assignments, as shown in Figure 3. This trend emphasises the crucial role of historical performance data in enhancing error prediction for Error-DKT.

Also, we observe the volume of submissions and the frequency of errors committed by students, which emerge as significant factors influencing model predictions due to the diverse and personalised strategies students employ. For example, problem 5 and 6 exhibits a significant decline in prediction accuracy, as highlighted in Figure 3 primarily due to their high error rates—twice and three times more than other problems, respectively (see Appendix A.2 Figure 10). In addition, by comparing the student-problem attempts in Figure 9 and Table 5 in Appendix A.2, we can see that problem 5 and 6 require many more attempts per student, and so appear to be different from the other questions. More attempts means we are again predicting the response to the compiler messages, and our predictive performance declines

In contrast, as shown in Figure 4, focused error prediction models (OE, IE) benefit from more error data, enabling these models to fine-tune their predictions more effectively compared to the damage they cause to the holistic model (HE). Furthermore, the analysis reveals that models perform better on problems that require similar skill sets in later stages (e.g., Problems 7, 9, and 10), suggesting that Error-DKT can successfully model students' knowledge of common error patterns.

Per Error Figure 5 shows the model struggling to accurately predict rare errors across different error classes. Nevertheless, an uptick in the models' ability to predict errors in classes 3 and 7, likely due to their frequent occurrence in problems 5 and 6 (see Appendix. A.1, Figure 7), suggests models like IE can benefit more. Additionally, errors 4 and 5, less common but often occurring with common error 1 (see Appendix. A.1, Figure 8), exhibit enhanced prediction accuracy. This indicates that models successfully extract insights from prevailing error patterns, thereby improving their predictive capabilities. We also note that the OKT models predominantly predicted the error class 2, "missing a return statement". This observation suggests that the estimated student knowledge level failed to prompt the LLM to generate codes incorporating previously unseen errors, such as those involving missing semicolons or unclosed curly brackets.

5.2 Correctness Prediction

Our methods focus on predicting individual errors, raising the question of whether these predictions can be aggregated into a holistic pass/fail assessment. According to the results in Table 4, this approach yields poorer performance compared to the original DKT method, which directly evaluates pass/fail outcomes. However, by incorporating the diverse input features outlined in Equation 1, we can significantly improve the correctness prediction

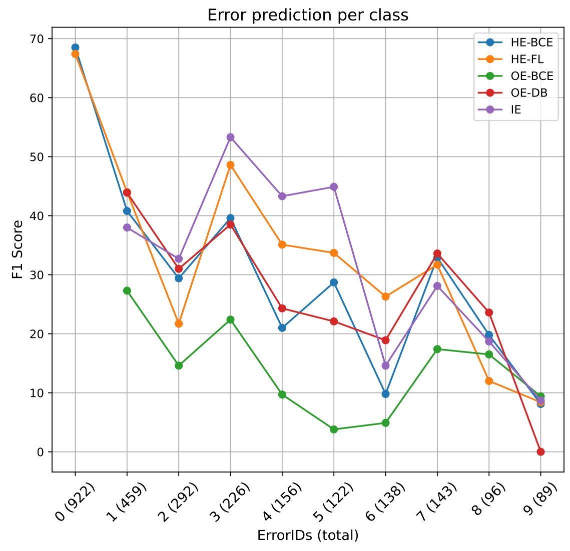

Figure 5: Model's Performance with various Prediction tasks per Error class.

capabilities of the original DKT model.

Model	**First**		**Overall**	
	AUC	F-beta	AUC	F-beta
Simple	46.7	40.9	50.9	58.1
Complex	58.0	46.0	61.6	67.2
OKT	n/a	30.8	n/a	43.0
HE-BCE	73.0	66.3	72.5	77.5
HE-FL	71.9	67.1	65.5	74.3
OE-BCE	68.0	60.5	63.4	70.1
OE-DB	68.7	60.8	68.7	75.1
IE	68.4	60.5	65.3	71.3
DKT	75.5	**72.7**	75.3	78.5
DKT*	**76.9**	72.3	**77.4**	**79.1**

Table 4: Model performance (AUC, F-beta) evaluation based on correctness (pass/fail) prediction. DKT* is trained using the new set of input features

5.3 Knowledge-driven prediction of students' submissions

The heatmaps presented in Figure 6 illustrate the capabilities and limitations of the Error-DKT model. The model exhibits proficiency in predicting errors that occur frequently but shows difficulty in identifying rarer errors. The effectiveness of the ensembled (two-step) approach is evident, as the accuracy of Step I predictions directly influences the subsequent error identification. For example, in Case 1, despite Step I yielding false positives, Step II strongly indicates the presence of errors, which are confirmed with ground truth values. This suggests the potential for alternative ensemble strategies that might allow Step II predictions to carry more weight. In contrast, Case 2 highlights that enhancing the accuracy of Step I predictions, which is generally more straightforward, could potentially lead to overall better performance in the model.

6 Conclusion and Future Works

In our study, we enhanced traditional knowledge tracing methods by developing a framework capable of predicting overall correctness and specific student errors. Our Error-DKT models demonstrated significant effectiveness, substantially outperforming baseline OKT models in overall attempts prediction with improvements of +15.8% and +23.2% using single-step and ensemble strategies (Holistic Error prediction), respectively. The ensemble approach significantly enhances accuracy by initially distinguishing error-free submissions from erroneous ones, and then specifically pinpointing the errors in submissions forecasted to fail.

Predictions for initial attempts generally exhibit higher accuracy, likely due to the uniformity of these submissions and the rich historical data available. However, as students revise their submissions in response to compiler feedback, the complexity of prediction increases, particularly for subsequent attempts. This issue is compounded in problems with high error rates and frequent submissions, like Problems 5 and 6, where performance notably declines. Despite the advantages in error prediction, our method showed less effectiveness in integrating individual errors into holistic pass/fail assessments compared to direct evaluations by traditional DKT methods. Nonetheless, the integration of diverse input features enhances the DKT model's ability to predict correctness. These findings underscore the potential of Error-DKT to improve the precision of error predictions and affirm the ongoing need for models that can adapt to complex error patterns and improve feedback mechanisms in educational settings.

For future work, several promising directions emerge. First, experimenting with advanced DKT architectures like AKT and DKVMN and refined ensemble methods may improve error prediction and accommodate a wider array of error types with more extensive datasets. Secondly, optimising the OKT model to extend its predictive competence to logical as well as compiler errors could yield more comprehensive error detection. Thirdly, there's

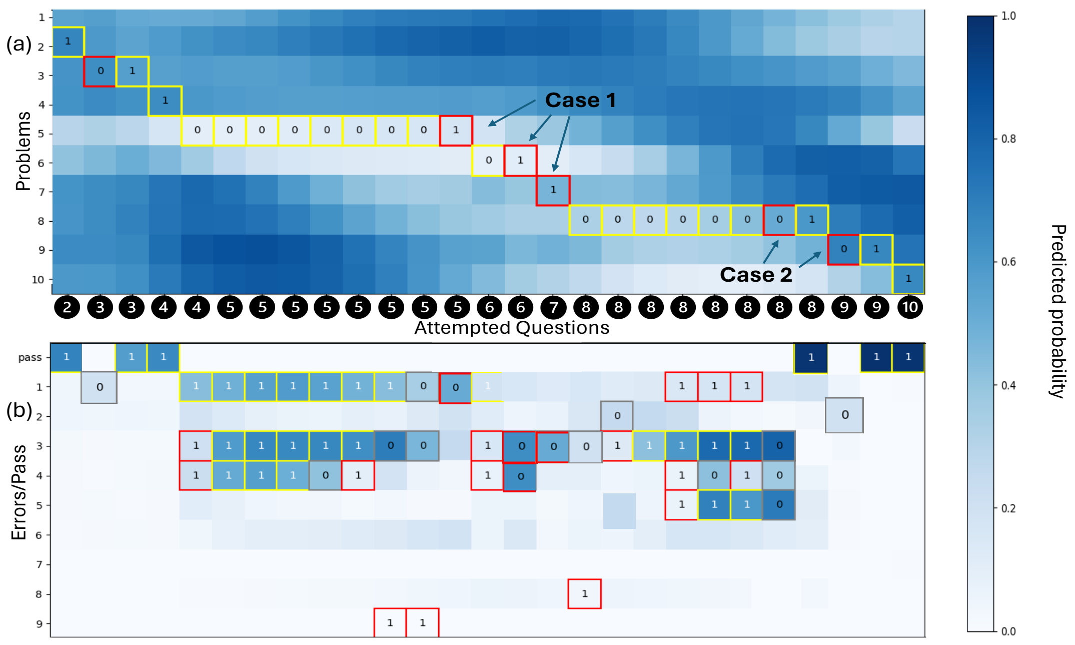

Figure 6: Ensembled Error-DKT prediction heatmap for a student over 27 attempts: (a) showcases Step I's correctness predictions, with yellow boxes indicating accurate predictions and red boxes highlighting incorrect predictions and the numbers in the cells represent the ground truth values, (b) displays specific error predictions using the Holistic approach, where red boxes with '1' signify undetected errors, and those with '0' indicate errors incorrectly predicted absent due to Step I's assessment. Grey boxes represent false error predictions.

a significant opportunity to enhance knowledge tracing models to interpret learned patterns, correlating them to specific knowledge areas, such as debugging skills reflected in students' coding progression. Finally, integrating our framework with an automated feedback mechanism will be vital in evaluating its effectiveness in delivering personalised, actionable feedback to students.

7 Limitations

Our study faces certain limitations. First, the modest performance of our Error-DKT models is partly due to the challenging prediction task and a small dataset (386 student summaries, referenced in Table 1). Despite this, Error-DKT shows promise in identifying specific student struggles better than baseline models. Second, we focus on a narrow dataset from one assignment and semester, limiting generalisation to wider programming contexts or error types. Given the novelty of this KT task, our concentration was solely on predicting compiler errors, with no examination of logical errors. This scope raises questions about the model's applicability across various programming scenarios. Lastly, using only DKT as a baseline for extending our approach may narrow our comparative analysis. Other current models like AKT, DKVMN could offer different insights or performance metrics. Nonetheless, our choice was informed by DKT's better performance to more recent deep models in related research (Shi et al., 2022; Liu et al., 2022), making it a logical starting point for exploring error predictions.

8 Acknowledgements

This work was conducted with the financial support of the Science Foundation Ireland Centre for Research Training in Artificial Intelligence under Grant No. 18/CRT/6223 and Grant No. 12/RC/2289-P2 at Insight the Science Foundation Ireland Research Centre for Data Analytics, which is co-funded under the European Regional Development Fund.

References

Ghodai Abdelrahman, Qing Wang, and Bernardo Nunes. 2023. Knowledge tracing: A survey. *ACM Comput. Surv*, 55.

Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29.

David Azcona, I-Han Hsiao, and Alan F Smeaton. 2019. Detecting students-at-risk in computer programming classes with learning analytics from students' digital footprints. *User Modeling and User-Adapted Interaction*.

Sahil Bhatia and Rishabh Singh. 2016. Automated correction for syntax errors in programming assignments using recurrent neural networks.

Albert T. Corbett and John R. Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4:253–278.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-balanced loss based on effective number of samples. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9260–9269.

Andrew Emerson, Fernando J. Rodríguez, Bradford Mott, Andy Smith, Wookhee Min, Kristy Elizabeth Boyer, Cody Smith, Eric Wiebe, and James Lester. 2019. Predicting early and often: Predictive student modeling for block-based programming environments. *International Educational Data Mining Society*.

J Figueiredo and F García-Peñalvo. 2021. Teaching and learning tools for introductory programming in university courses. pages 1–6.

Aritra Ghosh, Neil Heffernan, and Andrew S Lan. 2020. Context-aware attentive knowledge tracing. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2330–2339.

Aritra Ghosh, Jay Raspat, and Andrew Lan. 2021. Option tracing: Beyond correctness analysis in knowledge tracing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12748 LNAI:137–149.

Sumit Gulwani, Ivan Radiček, and Florian Zuleger. 2018. Automated clustering and program repair for introductory programming assignments. *ACM SIGPLAN Notices*, 53:465–480.

Zhenya Huang, Qi Liu, Chengxiang Zhai, Yu Yin, Enhong Chen, Weibo Gao, and Guoping Hu. 2019. Exploring multi-objective exercise recommendations in online education systems. *International Conference on Information and Knowledge Management, Proceedings*, pages 1261–1270.

Tanja Käser, Severin Klingler, Alexander G Schwing, and Markus Gross. 2017. Dynamic bayesian networks for student modeling. *IEEE Transactions on Learning Technologies*, 10(4):450–462.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2018. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327.

Naiming Liu, Zichao Wang, Richard Baraniuk, and Andrew Lan. 2022. Open-ended knowledge tracing for computer science education.

Qi Liu, Shuanghong Shen, Zhenya Huang, Enhong Chen, Senior Member, and Yonghe Zheng. 2023. A survey of knowledge tracing. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, v3.

Yunfei Liu, Yang Yang, Xianyu Chen, Jian Shen, Haifeng Zhang, and Yong Yu. 2020. Improving knowledge tracing via pre-training question embeddings. *IJCAI International Joint Conference on Artificial Intelligence*, 2021-January:1577–1583.

Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcao. 2019. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, 62.

Hiromi Nakagawa, Yusuke Iwasawa, and Yutaka Matsuo. 2019. Graph-based knowledge tracing: modeling student proficiency using graph neural network. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 156–163.

Sagar Parihar, Ziyaan Dadachanji, Praveen Kumar Singh, Rajdeep Das, Amey Karkare, and Arnab Bhattacharya. 2017. Automatic grading and feedback using program repair for introductory programming courses. *ITiCSE*.

Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas Guibas, Jascha Sohl-Dickstein, Stanford University, and Khan Academy. 2015. Deep knowledge tracing. *Advances in Neural Information Processing Systems*, 28.

Thomas W. Price, David Hovemeyer, Kelly Rivers, Ge Gao, Austin Cory Bart, Ayaan M. Kazerouni, Brett A. Becker, Andrew Petersen, Luke Gusukuma, Stephen H. Edwards, and David Babcock. 2020. Progsnap2: A flexible format for programming process data. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, pages 356–362.

Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. 2011. On the stratification of multi-label data. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part III 22*, pages 145–158. Springer.

Yang Shi, Min Chi, Tiffany Barnes, and Thomas W Price. 2022. Code-dkt: A code-based knowledge tracing model for programming tasks. *Proceedings of the 15th International Conference on Educational Data Mining*.

Dowon Song, Myungho Lee, and Hakjoo Oh. 2019. Automatic and scalable detection of logical errors in functional programming assignments. *Proceedings of the ACM on Programming Languages*, 3.

Ryo Suzuki, Gustavo Soares, Elena Glassman, Andrew Head, Loris D'Antoni, and Björn Hartmann. 2017. Exploring the design space of automatically synthesized hints for introductory programming assignments. *Conference on Human Factors in Computing Systems - Proceedings*, Part F127655:2951–2958.

Michael Thuné and Anna Eckerdal. 2019. Analysis of students' learning of computer programming in a computer laboratory context. *European Journal of Engineering Education*, 44:769–786.

L. Wang, Angela Sy, Larry Liu, and C. Piech. 2017. Learning to represent student knowledge on programming exercises using deep learning. *Educational Data Mining*.

Tong Wu, Qingqiu Huang, Ziwei Liu, Yu Wang, and Dahua Lin. 2020. Distribution-balanced loss for multi-label classification in long-tailed datasets. In *Computer Vision – ECCV 2020*, pages 162–178, Cham. Springer International Publishing.

Yingfei Xiong, Xinyuan Liu, Muhan Zeng, Lu Zhang, and Gang Huang. 2018. Identifying patch correctness in test-based program repair. *Proceedings - International Conference on Software Engineering*, pages 789–799.

Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 783–794.

Jiani Zhang, Xingjian Shi, Irwin King, and Dit-Yan Yeung. 2017. Dynamic key-value memory networks for knowledge tracing. In *Proceedings of the 26th international conference on World Wide Web*, pages 765–774.

A Appendix

A.1 Error Distribution

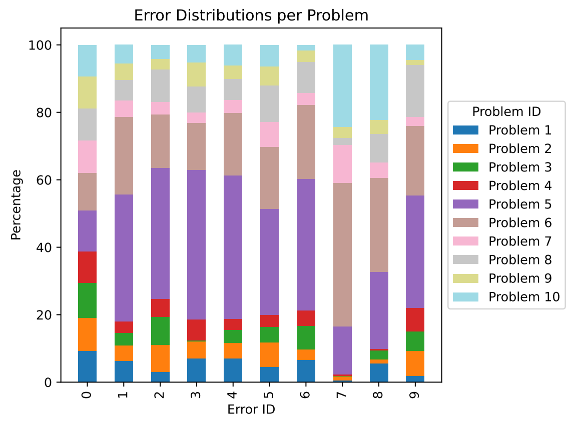

Figure 7: The percentage distribution of each question in the top ten error classes.

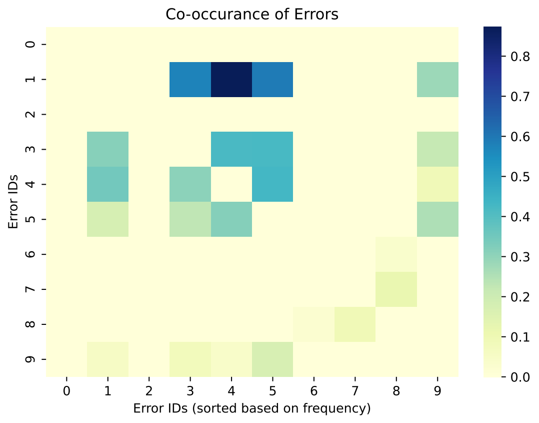

Figure 8: The heatmap showing the co-occurrence of the top ten errors.

Figure 7 maps out the primary distribution of errors across various problems, Figure 8 highlights an intricate aspect of this landscape: the co-occurrence of errors. This heatmap shows how frequently rarer errors appear alongside more common ones, offering insights into error correlations that can influence teaching strategies. Understanding these relationships is key to creating targeted interventions that simultaneously address multiple areas of student difficulty, thus streamlining the path to mastery and enhancing the overall efficacy of programming education.

A.2 Students Attempts

While nearly all students attempted the questions (see Figure 9), there was a notably higher number of attempts on questions five and six, with submissions averaging between 1700 to 2500 as highlighted in Table 5. This increase in attempts corresponded with a higher occurrence of errors in these questions (see Figure 10), suggesting that students were struggling to correct their mistakes, potentially encountering new challenges as they explored different solutions.

A.3 Model Architecture Configuration

Table 6 details the architecture and parameters that define the final models in our study. It encompasses the chosen input features, training configurations, loss functions, and the durations required for both training and inference across each model.

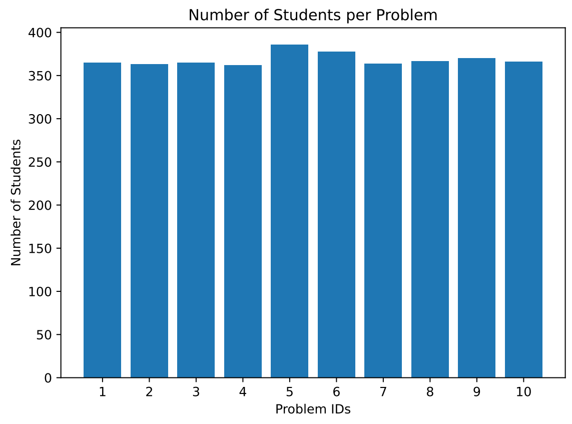

Figure 9: The total numbers of student attempting the 10 questions in assignment one.

Problem ID	Number of Attempts
1	663
2	694
3	699
4	653
5	2578
6	1743
7	678
8	852
9	635
10	800

Table 5: Number of Attempts per Problem

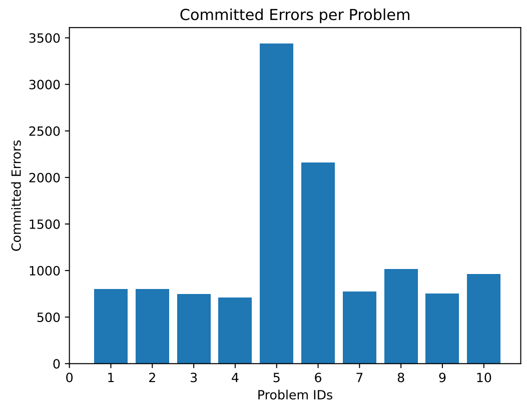

Figure 10: The number of errors committed in each student based on all the attempts.

Model	Input Features	Model Architecture	Output	Training Configurations	Train and inference Time
Simple Baseline	Top errors per problem	-	10	-	1m36s
Complex Baseline	Top errors per problem per attempts	-	10	-	1m36s
HE-BCE	$\mathrm{E_r}(r_t)$, $\mathrm{E_e}(e_t)$, $\mathrm{E_c}(c_t)$	layers=1, hidden=512	10	lr=0.00073, epochs=28, bs=16	7m44s
HE-FL	$\mathrm{E_r}(r_t)$, $\mathrm{E_e}(e_t)$, $\mathrm{E_p}(p_t)$	layers=1, hidden=256	10	lr=0.0004, epochs=70, α(Fl)=0.96, γ(FL)=2.31, bs=16	8m14s
OE-DBloss	$\mathrm{E_r}(r_t)$, $\mathrm{E_e}(e_t)$, $\mathrm{E_{\{ref\}}}(\{ref\}_t)$	layers=1, hidden=256	9	lr=0.000485, epochs=80, α(FL)=0.96, γ(FL)=4.82, β(CB)=0.955, α(DB)=0.93, γ(DB)=0.89, β(DB)=1.53, bs=16	14m14s
OE-BCE	$\mathrm{E_r}(r_t)$, $\mathrm{E_e}(e_t)$, $\mathrm{E_{\{ref\}}}(\{ref\}_t)$	layers=1, hidden=256	9	lr=0.0009, epochs=80, bs=16	10m44s
IE-BCE	$\mathrm{E_r}(r_t)$, $\mathrm{E_e}(e_t)$, $\mathrm{E_{\{ref\}}}(\{ref\}_t)$	layers=1, hidden=256	9	lr=0.0009, epochs=80, bs=16	25m32s
DKT	$\mathrm{E_r}(r_t)$	layers=1, hidden=512	1	lr=0.0005, epochs=50, bs=16	8m35s
OE-BCE*	$\mathrm{E_r}(r_t)$, $\mathrm{E_e}(e_t)$, $\mathrm{E_p}(p_t)$	layers=2, hidden=256	9	lr=0.00071, epochs=70, bs=16	6m32s

Table 6: Model architecture configurations for various prediction task and the set of best input features.