Personalized Programming Guidance Based on Deep Programming Learning Style Capturing

Yingfan Liu[1], Renyu Zhu[1,3], and Ming Gao[1,2(✉)]

[1] School of Data Science and Engineering,
East China Normal University, Shanghai, China
yfliu.ds@stu.ecnu.edu.cn, mgao@dase.ecnu.edu.cn
[2] KLATASDS-MOE in School of Statistics,
East China Normal University, Shanghai, China
[3] NetEase Fuxi AI Lab, Hangzhou, Zhejiang, China
zhurenyu@corp.netease.com

Abstract. With the rapid development of big data and AI technology, programming is in high demand and has become an essential skill for students. Meanwhile, researchers also focus on boosting the online judging system's guidance ability to reduce students' dropout rates. Previous studies mainly targeted at enhancing learner engagement on online platforms by providing personalized recommendations. However, two significant challenges still need to be addressed in programming: ***C1***) how to recognize complex programming behaviors; ***C2***) how to capture intrinsic learning patterns that align with the actual learning process. To fill these gaps, in this paper, we propose a novel model called **P**rogramming **E**xercise **R**ecommender with Learning **S**tyle (**PERS**), which simulates learners' intricate programming behaviors. Specifically, since programming is an iterative and trial-and-error process, we first introduce a positional encoding and a differentiating module to capture the changes of consecutive code submissions (which addresses ***C1***). To better profile programming behaviors, we extend the Felder-Silverman learning style model, a classical pedagogical theory, to perceive intrinsic programming patterns. Based on this, we align three latent vectors to record and update programming ability, processing style, and understanding style, respectively (which addresses ***C2***). We perform extensive experiments on two real-world datasets to verify the rationality of modeling programming learning styles and the effectiveness of PERS for personalized programming guidance.

Keywords: Programming Education · Sequential Recommendation · Learning Style

1 Introduction

The rapid advancement of AI technology has profoundly influenced on individuals of diverse backgrounds and skill levels. In this connection, online judge

W. Hong and G. Kanaparan (Eds.): ICCSE 2023, CCIS 2023, pp. 214–231, 2024.
https://doi.org/10.1007/978-981-97-0730-0_20

systems have emerged as an indispensable avenue for those seeking to boost their programming proficiency[1]. However, despite the growing popularity of this learning modality, high dropout rates have been observed, attributable to the inadequate provision of personalized instructions tailored to learners' unique learning preferences [22].

Recently, recommender systems have been widely applied in online education scenarios to facilitate personalized learning. There are various recommendation models, including CF(collaborative filtering)-based methods [20,31], content-based methods [1,13] and deep-learning-based methods [7,8]. These general models aim to provide personalized recommendations by capturing users' interests and needs through static preferences and individual interactions. In the context of programming, however, the learning process exhibits a dynamic and progressive nature. This represents an essential application of the sequential recommendation (SR) task, which predicts subsequent behavioral sequences based on historical records [12,27,29,32].

While extant SR models have yielded successful results in e-learning contexts, there remain significant gaps in directly deploying them to programming scenarios [18]. As illustrated by Fig. 1, programming learning differs from traditional learning in two crucial respects: i) it enables learners to make multiple attempts on the same exercise and edit their previous submissions based on the feedback received from the compiler and, ii) the platform can record fine-grained behavioral data related to programming, including code snippets, compilation time, and compilation status. Furthermore, current sequential models prioritize learners' patterns with little regard to their intrinsic behaviors, including learning styles. These styles reflect the ways in which learners process and comprehend information, and are thus factors that cannot be ignored.

Consequently, the study of SR in programming learning confronts two significant challenges. First, it is imperative to model distinctive and fine-grained patterns involved in programming, including code-related side features and iterative submission behavior (***C1***). Second, there is an urgent need to incorporate pedagogical theory into the model to bolster its interpretability with the actual learning process (***C2***).

To address the above challenges, we propose a new model named **P**rogramming **E**xercise **R**ecommender with Learning **S**tyle (PERS). To simulate the iterative process in programming, we employ a two-step approach. Firstly, we map programming exercises and code-related features (such as code snippets, execution time, and execution status) into embeddings using a representation module with positional encoding. Secondly, we formulate a differentiating module that calculates the changes between consecutive code submissions. This module can adeptly capture fine-grained learning patterns by effectively distinguishing between intra-exercise or inter-exercise attempts (for ***C1***). To enhance the consistency between our proposed model and the actual learning process, we draw

[1] A theory proposed by Richard M. Felder and Linda K. Silverman to describe individuals' preferred ways of learning. Further details will be provided in the related work section.

inspiration from a pedagogical theory known as the Felder-Silverman Learning Style Model (FSLSM) [4], which is widely utilized in educational scenarios for mining learning patterns and delivering personalized guidance. Considering the processing and understanding dimensions in FSLSM, we present a formal definition and detailed descriptions of programming learning styles in this paper. On this foundation, we develop three latent vectors: programming ability, processing style, and understanding style, which are designed to track the learners' intrinsic behavioral patterns during the programming process (for ***C2***). After obtaining the above vectors, our model employs a multilayer perceptron (MLP) to generate personalized predictions that align individuals' learning preferences. In summary, the main contributions of this paper are summarized as follows:

- Our study endeavors to furnish personalized programming guidance by emulating the iterative and trial-and-error programming learning process, thereby offering a novel vantage point on programming education.
- We have meaningfully incorporated the FSLSM pedagogical theory into our model, enabling us to effectively capturing the intrinsic behavioral patterns of students while also enhancing rationality and consistency.
- We conduct experiments on two real-world datasets to validate the efficacy and interpretability of our approach.

2 Related Works

2.1 Sequential Recommendation

Sequential recommendation models aim to incorporate users' personalized and contextual information based on their historical interactions [16] to predict the future behaviors.

In earlier studies, researchers considered Markov chains as a powerful method to model interaction processes and capture users' sequential patterns [2,9]. Later, the advent of recurrent neural networks (RNN) greatly has expanded the potential of recommender systems to process multi-type input data and understand complex item transitions. For example, [10] first adopt RNN on real-life session-based recommendations and then enhance the backbone with parallel RNN to leverage more rich features [11]. There are various techniques designed to improve the RNN-based models, such as data augmentation (GRU4Rec [29]), data reconstruction (SRGNN [32]), and unified module injection (SINE [28], CORE [12]). Recently, another line of works has seeked to use the Transformer module to capture global information, which RNN overlooks. For instance, BERT4Rec [27] utilize bidirectional self-attention with Cloze tasks during training to enhance the hidden representation.

2.2 Sequential Recommendation in E-Learning

Existing research on SR in e-learning typically focus on recommending the most appropriate resources, such as courses and exercises, to learners by capturing

their static and dynamic characteristics through their past behavioral record [14]. For instance, [15] and [19] propose a cognitive diagnostic method to model students' proficiency on each exercises based on probabilistic matrix factorization and students' proficiency. [25] apply a knowledge tracing model with an enhanced self-attention to measures students' mastery states and assist model to recommend. These methods effectively capture students' preferences and mastery of knowledge points. However, they often overlook the impact of students' internal learning styles.

In the field of programming, some preliminary attempts have been made to explore the personalized recommendations. For example, [18] apply BERT [3] to encode students' source code and propose a knowledge tracing model to capture mastery of programming skills. However, the dynamic sequential patterns in existing works are not consistent with real programming process due to ignore the iterative process.

2.3 Learning Style Model

Learning styles refer to the way in which students prefer to obtain, process and retain information [5]. The most common theoretical models include Felder-Silverman Learning Style Model (FSLSM), Kolb's learning style [17] and VARK model [23]. Previous research has demonstrated that the FSLSM is more comprehensible and appropriate for identifying learning styles in online learning compared to other models [21]. This model describes learning styles from four dimensions: the perspective of perception (sensitive/intuitive), information input (visual/verbal), processing (active/reflective) and understanding (sequential/global) based on the learner's behavior patterns during learning process.

3 Preliminaries

3.1 Programming Learning Style Model

Inspired by the FSLSM, we define a programming learning style model (PLSM) centered around the problem-solving behavior observed in online judging systems.

As shown in Table 1, the PLSM delineates the inherent learning patterns during programming through two dimensions: processing and understanding. In terms of processing, learners can be classified as either active or reflective. When solving exercises, active learners tend to think through a complete answer before submitting their solution, while reflective learners prefer to attempt the same exercise multiple times and refine their previous submissions based on the compiler feedback. As for the dimension of understanding, learners can be labeled as sequential or global. Sequential learners tend to approach learning tasks in a progressive sequence, such as in numerical or knowledge concept order. In contrast, global learners tend to approach tasks in a non-linear fashion, such as by selecting tasks that they find most interesting or engaging. These distinct learning styles reflect learners' preferences and can significantly impact the trajectory of their problem-solving process.

Table 1. Programming Learning Style Model

Dimension	Label	Learning Characteristics
Processing	Active	Solving exercises by figuring out a complete answer before submitting
	Reflective	Solving exercises through multiple trials and errors
Understanding	Sequential	Solving step-by-step
	Global	Solving by leaps and bounds

3.2 Problem Definition

To foster learners' involvement and enhance their programming skills in online judge systems, we present a new task called programming exercise recommendation (PER). The definition of PER is as follows and an example of data model is depicted in Fig. 1.

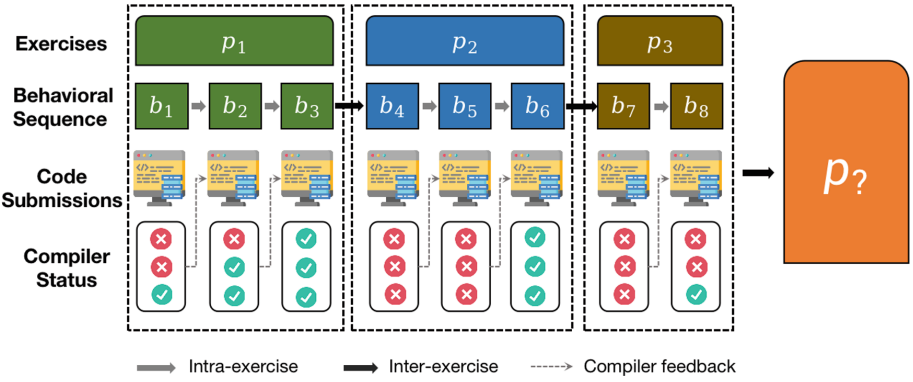


Fig. 1. Data model for PER task

Definition: Programming Exercise Recommendation. Suppose there are n online learning users $\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$ with problem-solving behavior logs $\mathcal{B} = \{B_1, B_2, \cdots, B_n\}$ and m programming exercises $\mathcal{P} = \{p_1, p_2, \cdots, p_m\}$. Specifically, the i-th record $B_i = \{b_1, b_2, \cdots, b_{l_i}\}$ represents the interaction sequence of the i-th learner u_i, where l_i represents the length of the sequence. Each element b_j in the sequence is a triple $\langle p_{b_j}, c_{b_j}, r_{b_j} \rangle$ consisting of the problem p_{b_j}, the code c_{b_j} and the compilation result r_{b_j}. The ultimate goal of programming exercise recommendation is to predict learners' learning preferences in the future based on the past interaction behavior B_i between learners and exercises. that is, the next exercise $p_{b_{li}+1}$ that will be tried. Correspondingly, in machine learning methods, the optimization objective is:

$$l_i = \max_{\mathcal{A}} \prod_{(b_i, p_{b_{l_i+1}}) \in \mathcal{B} \cup \mathcal{B}^-} \log \mathcal{A}(p_{b_{l_i+1}}|B_i)^{y_i} (1 - \mathcal{A}(p_{b_{l_i+1}})|B_i)^{(1-y_i)}, \quad (1)$$

where $\mathcal{A}$ is a probabilistic prediction model, such as neural networks, whose output is to predict the probability of interacting with the next exercise $p_{b_{li}+1}$ based on the historical behavior sequence B_i. $\mathcal{B}^-$ is a set of negative samples, i.e., the exercises that learner u_i has not interacted with label $y_i = 1$ if and only if $(B_i, p_{b_{l_i+1}}) \in \mathcal{B}$, otherwise $y_i = 0$.

4 PERS Framework

In this section, we propose a deep learning framework, namely PERS, to solve programming exercise recommendation. As shown in Fig. 2, the architecture of PERS is mainly composed of four functional modules: representing, differentiating, updating and predicting. The details of the four modules are given in the following.

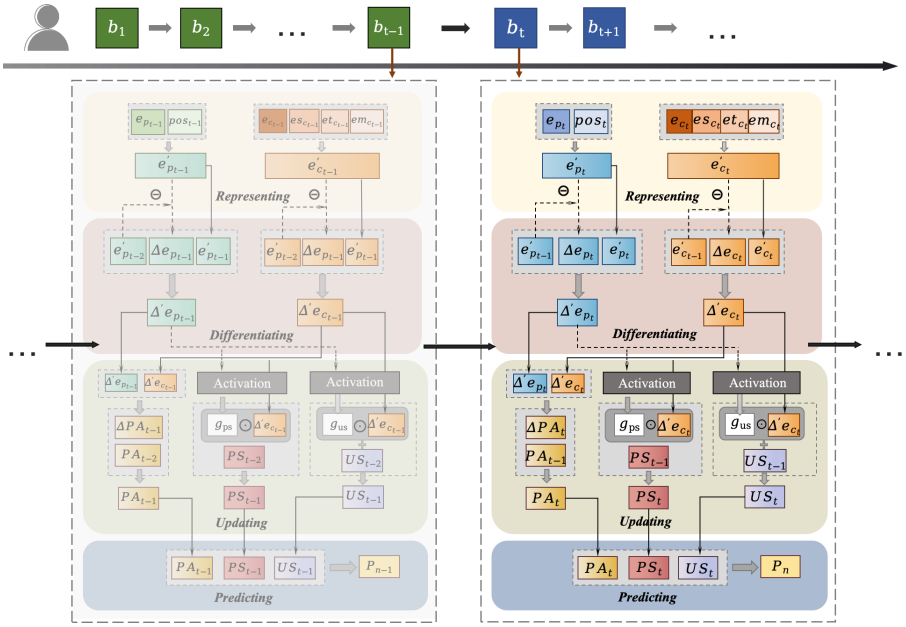


Fig. 2. PERS Architecture

4.1 Representing Module

The representing module mainly focuses on obtaining the embedding of the two inputs: exercises and codes.

Exercise Representation. As demonstrated in Fig. 1, learners typically attempt a programming exercise multiple times until they pass all test cases. Even when trying the same exercise, the compilation results for each attempt are distinct and progressive. Therefore, the exercise embedding and the positional embedding in the sequence are both critical. Suppose p_t denotes the programming exercise coded by the learners at time t. First, we use a projection matrix $\mathbf{E}_p \in \mathcal{R}^{(N+2)\times d_p}$ to represent each exercise by its id, where N is the total number of exercises and d_p is the dimension. The first dimension of the projection matrix here is $N+2$ because two zero pads are added. Then the representation vector for the problem p_t can be obtained as $\mathbf{e}_{p_t} \in \mathcal{R}^{d_p}$. In addition, inspired by the work [30], we use the sinusoidal function to acquire the position embedding $\mathbf{pos}_t$ at time t:

$$\mathbf{pos}_{(t,2i)} = \sin(t/10000^{2i/d_{pos}}), \tag{2}$$

$$\mathbf{pos}_{(t,2i+1)} = \cos(t/10000^{2i/d_{pos}}). \tag{3}$$

where d_{pos} denotes the dimension. Based on the exercise embedding $\mathbf{e}_{p_t}$ and the position embedding $\mathbf{pos}_t$, we obtain the enhanced exercise embedding $\mathbf{e}'_{p_t}$ through an MLP:

$$\mathbf{e}'_{p_t} = \mathbf{W}_1^T[\mathbf{e}_{p_t} \oplus \mathbf{pos}_t] + \mathbf{b}_1, \tag{4}$$

where $\oplus$ denotes the vector concatenation, $\mathbf{W}_1 \in \mathcal{R}^{(d_p+d_{pos})\times d_k}$, $\mathbf{b}_1 \in \mathcal{R}^{d_k}$ are learnable parameters.

Code Representation. Suppose c_t denotes the code submitted by the learners at time step t. First, we apply a code pre-training model CodeBERT [6] to obtain the initial embedding of code $\mathbf{e}_{c_t} \in \mathcal{R}^{d_c}$. Additionally, we employ different projection matrices to obtain the representation vectors of code-related side features: the execution time $\mathbf{et}_{c_t} \in \mathcal{R}^{d_{ct}}$, the execution memory $\mathbf{em}_{c_t} \in \mathcal{R}^{d_{cm}}$, and the execution status $\mathbf{es}_{c_t} \in \mathcal{R}^{d_{cs}}$. After all the representation vectors are generated, we can obtain the enhanced code embedding $\mathbf{e}'_{c_t}$ by an MLP:

$$\mathbf{e}'_{c_t} = \mathbf{W}_2^T[\mathbf{e}_{c_t} \oplus \mathbf{es}_{c_t} \oplus \mathbf{et}_{c_t} \oplus \mathbf{em}_{c_t}] + \mathbf{b}_2, \tag{5}$$

where $\mathbf{W}_2 \in \mathcal{R}^{(d_c+d_{cs}+d_{ct}+d_{cm})\times d_k}$ is the weight matrix, $\mathbf{b}_2 \in \mathcal{R}^{d_k}$ is the bias term.

4.2 Differentiating Module

As the introduction highlights, one of the challenges in PER is to simulate the iterative and trial-and-error process of programming learning. In this paper, we develop a differentiating module to capture fine-grained learning patterns. To distinguish whether students are answering the same exercise or starting a new one, we first calculate the exercise difference embedding $\Delta\mathbf{e}_{p_t}$ between

students' present exercise embedding $\mathbf{e}_{p_t}^{'}$ and previous exercise embedding $\mathbf{e}_{p_{t-1}}^{'}$ by subtraction. Then, we feed the above three embeddings into a multi-layer perceptron to output the final exercise difference embedding $\Delta^{'}\mathbf{e}_{p_t}$:

$$\Delta\mathbf{e}_{p_t} = \mathbf{e}_{p_t}^{'} - \mathbf{e}_{p_{t-1}}^{'} \tag{6}$$

$$\Delta^{'}\mathbf{e}_{p_t} = \mathbf{W}_3^T[\Delta\mathbf{e}_{p_t} \oplus \mathbf{e}_{p_t}^{'} \oplus \mathbf{e}_{p_{t-1}}^{'}] + \mathbf{b}_3, \tag{7}$$

For the same exercise, the codes students submit are different at each attempt, which can indicate their progress in the trial-and-error process. Therefore, we use students' present code embedding $\mathbf{e}_{c_t}^{'}$, previous code embedding $\mathbf{e}_{c_{t-1}}^{'}$ and the difference between them $\Delta\mathbf{e}_{c_t}$ to obtain the final code difference embedding $\Delta^{'}\mathbf{e}_{c_t}$:

$$\Delta\mathbf{e}_{c_t} = \mathbf{e}_{c_t}^{'} - \mathbf{e}_{c_{t-1}}^{'}, \tag{8}$$

$$\Delta^{'}\mathbf{e}_{c_t} = \mathbf{W}_4^T[\Delta\mathbf{e}_{c_t} \oplus \mathbf{e}_{c_t}^{'} \oplus \mathbf{e}_{c_{t-1}}^{'}] + \mathbf{b}_4, \tag{9}$$

where $\mathbf{W}_3, \mathbf{W}_4 \in \mathcal{R}^{3d_k \times d_k}$ is the weight matrix, $\mathbf{b}_3, \mathbf{b}_4 \in \mathcal{R}^{d_k}$ is the bias term.

4.3 Updating Module

The purpose of this module is to update the latent states that represent the learner's intrinsic learning style. Inspired by the classic learning style model FSLSM, we propose two hidden vectors, processing style $\mathbf{PS}_t$ and understanding style $\mathbf{US}_t$, to capture the programming learning style of learners. In addition, motivated by the programming knowledge tracing research [33], we introduce another hidden vector called programming ability $\mathbf{PA}_t$ to enhance the modeling of programming behavior.

First, we assume that all learners start with the same programming ability $\mathbf{PA}_0$, and their programming ability will gradually improve as they progress through exercises. The learners' programming ability $\mathbf{PA}_t$ at time step t depends on their performance in completing the current exercises as well as their previous programming ability $\mathbf{PA}_{t-1}$. The corresponding update process is as follows:

$$\Delta\mathbf{PA} = \mathbf{W}_5^T[\mathbf{e}_{p_t}^{'} \oplus \mathbf{e}_{c_t}^{'}] + \mathbf{b_5}, \tag{10}$$

$$\mathbf{PA}_t = \mathbf{W}_6^T[\Delta\mathbf{PA} \oplus \mathbf{PA}_{t-1}] + \mathbf{b_6}, \tag{11}$$

where $\mathbf{W}_5, \mathbf{W}_6 \in \mathcal{R}^{2d_k \times d_k}$ are weight matrices, $\mathbf{b}_5, \mathbf{b}_6 \in \mathcal{R}^{d_k}$ are bias terms. When $t = 0$, $\mathbf{PA}_0 \in \mathcal{R}^{d_k}$ is initialized as a vector of all zeros.

Similarly, the initial processing style $\mathbf{PS}_0 \in \mathcal{R}^{d_k}$ at time $t = 0$ is also initialized as a vector of all zeros. As shown in Table 1, the learner's processing style mainly manifests in their continuous trial-and-error behavior on the same exercise. Leveraging the difference of exercise $\Delta^{'}\mathbf{e}_{p_t}$ and code $\Delta^{'}\mathbf{e}_{c_t}$ generated from the difference module, we introduce a gating mechanism to update the learner's processing style vector $\mathbf{PS}_t$. We first calculate a selection gate $\mathbf{g}_{ps}$ using $\Delta^{'}\mathbf{e}_{p_t}$,

which determines whether the current exercise is identical to the previous one. Then $\mathbf{g}_{ps}$ is multiplied by $\Delta^{'}\mathbf{e}_{c_t}$ to figure out how much semantic information should be learned from the code. Finally, we concatenate the result with the previous processing style $\mathbf{PS}_{t-1}$ and employ a multi-layer perception to fuse these vectors as follows:

$$\mathbf{g}_{ps} = \tanh(\mathbf{W}_7^T \Delta^{'}\mathbf{e}_{p_t} + \mathbf{b}_7), \tag{12}$$

$$\mathbf{PS}_t = \mathbf{W}_8^T[\mathbf{PS}_{t-1} \oplus (\mathbf{g}_{ps} \odot \Delta^{'}\mathbf{e}_{c_t})] + \mathbf{b}_8, \tag{13}$$

where tanh is the non-linear activation function, $\mathbf{W}_7 \in \mathcal{R}^{d_k \times d_k}$, $\mathbf{W}_8 \in \mathcal{R}^{2d_k \times d_k}$, $\mathbf{b}_7, \mathbf{b}_8 \in \mathcal{R}^{d_k}$ are trainable parameters, $\odot$ is the vector element-wise product operation.

Another latent vector is the understanding style $\mathbf{US}_t$, which indicates whether learners prefer to learn step-by-step or in leaps and bounds. It is derived from the learner's historical records. Thus, the initial understanding style $\mathbf{US}_0 \in \mathcal{R}^{d_k}$ is also initialized as a vector of zeros. Similar to the processing style, we also employ a gating mechanism to determine whether the learner is attempting the same exercise, and subsequently update the current understanding style $\mathbf{US}_t$ based on the previous one $\mathbf{US}_{t-1}$:

$$\mathbf{g}_{us} = \tanh(\mathbf{W}_9^T \Delta^{'}\mathbf{e}_{p_t} + \mathbf{b}_9), \tag{14}$$

$$\mathbf{US}_t = \mathbf{US}_{t-1} + \mathbf{W}_{10}^T(\mathbf{g}_{us} \odot \mathbf{e}_{p_t}^{'}) \tag{15}$$

where $\mathbf{W}_9, \mathbf{W}_{10} \in \mathcal{R}^{d_k \times d_k}$ are weight matrices, $\mathbf{b}_9 \in \mathcal{R}^{d_k}$ is the bias term.

4.4 Predicting Module

After obtaining the learner's programming ability $\mathbf{PA}_t$, processing style $\mathbf{PS}_t$, and understanding style $\mathbf{US}_t$, we can predict the next exercise in the predicting module. First, the three intrinsic vectors are concatenated and then projected to the output layer using a fully connected network to get $\mathbf{Pre}_t$. After that, we encode $\mathbf{Pre}_t$ into an m-dimensional project matrix and obtain the final probability vector $\mathbf{p}_n$ of exercises being recommended at the next step.

$$\mathbf{Pre}_t = \mathbf{W}_{11}^T[\mathbf{PL}_t \oplus \mathbf{PS}_t \oplus \mathbf{US}_t] + \mathbf{b}_{11}, \tag{16}$$

$$\mathbf{p}_n = \mathbf{W}_{12}^T \mathbf{Pre}_t + \mathbf{b}_{12} \tag{17}$$

where $\mathbf{W}_{11} \in \mathcal{R}^{3d_k \times d_k}$ and $\mathbf{W}_{12} \in \mathcal{R}^{d_k \times d_n}$ are weight matrices, $\mathbf{b}_{11} \in \mathcal{R}^{d_k}$ and $\mathbf{b}_{12} \in \mathcal{R}^{d_m}$ is the bias term.

5 Experiments

In this section, we aim to evaluate the effectiveness of PERS on programming exercise recommendation through empirical evaluation and answer the following research questions:

- **RQ1**: How does PERS perform compared with state-of-the-art pedagogical methods and sequential methods on programming exercise recommendation?
- **RQ2**: What is the impact of different components on the performance of PERS ?
- **RQ3**: How do the primary hyperparameters influence the performance of our model?
- **RQ4**: Can the proposed method learn meaningful intrinsic representations of students during programming?

5.1 Experimental Settings

Datasets. We evaluate our proposed method PERS on two real-world datasets: BePKT [33]and CodeNet [24]. The two datasets are both collected from online judging systems, including problems, codes and rich contextual information such as problem descriptions and code compilation results. Due to the millions of behaviors and contextual data in CodeNet, memory overflow exists when processing contextual features such as code and problem descriptions. Therefore, we sample the CodeNet dataset based on sequence length and submit time, resulting in two smaller-scale datasets: CodeNet-len and CodeNet-time. A brief overview of each dataset is listed as follows:

- **BePKT:** collected from an online judging system[2] targeted at university education, with its users primarily being college students who start learning to program.
- **CodeNet:** collected and processed by IBM researchers from two large-scale online judgment systems AIZU[3] and AtCoder[4]. The dataset contains hundreds of thousands of programming learners from different domains.
- **CodeNet-len:** a subset of the CodeNet dataset, which only keeps learners' programming behavioral sequences with lengths between 500 and 600.
- **CodeNet-time:** a subset from the CodeNet dataset with submission timestamps between March and April 2020.

Table 2 presents detailed statistics for the above datasets. Specifically, the calculation formula of #Sparsity is as follows:

$$\#\text{Sparsity} = 1 - \frac{\#\text{Interactions}}{\#\text{Students} \times \#\text{Exercises}}, \tag{18}$$

[2] https://judgefield.shuishan.net.cn/.
[3] https://onlinejudge.u-aizu.ac.jp/home.
[4] https://atcoder.jp/.

Table 2. Detailed statistics of all datasets in experiments, where #Learners denotes the number of learners, #Interactions denotes the number of interactions, #Exercises denotes the number of exercises, #Sparsity denotes the sparsity of the dataset, #Pass-Rate denotes the proportion of successful submissions in all submissions, and #APE(short for Avg-Attempts-Per-Exercise) denotes the average number of attempts on the same programming exercise.

Dataset	#Learners	#Interactions	#Exercises	#Sparsity	#Pass-Rate	#APE
BePKT	907	75,993	553	84.85%	32.03%	3.18
CodeNet	154,179	13,916,868	4,049	97.77%	53.61%	2.05
CodeNet-time	26,270	811,465	2,465	98.75%	53.42%	1.89
CodeNet-len	1,107	605,661	3,308	83.46%	56.88%	1.87

Baselines. We compare PERS with the following 8 comparable baselines, which can be grouped into two categories:

- **Pedagogical methods: ACKRec** [8] and **LPKT** [26] are two representative methods in e-learning recommendation. ACKRec constructs a heterogeneous information network to capture entity relationships. LPKT develops a model by simulating students' learning processes.
- **Sequential methods:** We introduce 6 state-of-the-art sequential models, which are 1) **GRU4Rec** [29] introduces data augmentation on recurrent neural network to improve model performance. 2) **GRU4Recf** [11] further integrates a parallel recurrent neural network to simultaneously represent clicks and feature vectors within interactions. 3) **BERT4Rec** [27] introduces a two-way self-attention mechanism based on BERT [3]. 4) **SRGNN** [32] converts user behavior sequences into graph-structured data and introduces a graph neural network to capture the relationship between items. 5) **SINE** [28] proposes a sparse interest network to adaptively generate dynamic preference. 6) **CORE** [12] designs a representation consistency model to pull the vectors into the same space.

Since all the above baselines do not incorporate code as the model input, for a fair comparison, we implement a degraded version of PERS:

- **ERS**: Remove the code feature input and all subsequent related modules in PERS.

Evaluation Metrics. To fairly compare different models, inspired by the previous [8], we choose the HR@10 (Hit Ratio), NDCG@10 (Normalized Discounted Cumulative Gain), and MRR@10 (Mean Reciprocal Rank) as the evaluation metrics.

Table 3. The overall performance on two full datasets and two sample datasets. OOM refers to out of memory.

Datasets	Metrics	Pedegogicals		Sequentials						Ours	
		LPKT	ACKRec	GRU4Rec	GRU4Recf	BERT4Rec	SRGNN	SINE	CORE	ERS	PERS
BePKT	HR@10	0.8762	0.8849	0.9172	0.9135	0.7369	0.9074	0.6419	0.9172	0.9256	**0.9288**
	MRR@10	0.6743	0.6838	0.7057	0.7053	0.5324	0.6870	0.4059	0.6923	0.7104	**0.7153**
	NDCG@10	0.7128	0.7269	0.7573	0.7560	0.5816	0.7408	0.4623	0.7466	0.7645	**0.7688**
CodeNet	HR@10	0.8423	0.8372	0.8728	OOM	0.6715	0.8700	0.7276	0.8566	**0.8803**	OOM
	MRR@10	0.6372	0.6305	0.6927	OOM	0.4205	0.6879	0.4643	0.5934	**0.7012**	OOM
	NDCG@10	0.6983	0.6847	0.7374	OOM	0.4811	0.7332	0.5285	0.6581	**0.7435**	OOM
CodeNet-len	HR@10	0.7539	0.7754	0.7812	0.7767	0.1157	0.7821	0.4602	0.7865	0.7934	**0.8010**
	MRR@10	0.5816	0.5938	0.6189	0.6143	0.0453	0.6073	0.2411	0.5251	0.6235	**0.6322**
	NDCG@10	0.6043	0.6139	0.6590	0.6545	0.0614	0.6504	0.2934	0.5850	0.6631	**0.6704**
CodeNet-time	HR@10	0.8532	0.8425	0.8993	0.8991	0.7517	0.8989	0.7742	0.8962	0.9058	**0.9167**
	MRR@10	0.6931	0.6852	0.7309	0.7316	0.6473	0.7236	0.5458	0.6060	0.7422	**0.7574**
	NDCG@10	0.7348	0.7233	0.7731	0.7735	0.5978	0.7675	0.6016	0.6775	0.7841	**0.7923**

Training Details. For pedagogical methods, we use the original codes released by their authors[5][6]. Additionally, we implement the PERS model and other baseline models using PyTorch and the RecBole library[7]. We run all the experiments on a server with 64 G memory and two NVIDIA Tesla V100 GPUs. For all models, we set the max sequence length to 50, the batch size of the training set to 2048 and the test set to 4096, and the optimizer to Adam. For the PERS model, we set the exercise and code representation embedding dimensions to 128. We perform the hyper-parameter tuning for the learning rate $\{0.1, 0.01, 0.001\}$, the layer number $\{1, 2, 3\}$, and the dropout rate $\{0.1, 0.3, 0.5\}$. For all methods, we fine-tune the hyperparameters to achieve the best performance and run experiments three times to report the average results.

5.2 RQ1: Overall Performance

Table 3 summarizes the performance results. We evaluate the methods on four datasets under three evaluation metrics. The best results are highlighted in bold and the best baselines are underlined. From results in the Table 3, we make the following observations:

- Our proposed models, PERS and ERS, demonstrate state-of-the-art performance on large-scale programming learning datasets. For instance, in the case of the CodeNet dataset, our models exhibit a significant improvement of 1.41% on HR@10, 1.30% on MRR@10, 1.12% on NDCG@10 over the best baseline.
- Code features can significantly improve the performance of the model. In the BePKT, CodeNet-len, and CodeNet-time datasets, the PERS model with

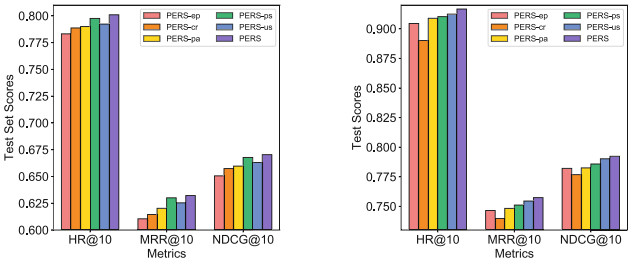
[5] https://github.com/JockWang/ACKRec.
[6] https://github.com/bigdata-ustc/EduKTM/tree/main/EduKTM.
[7] https://recbole.io/.

code features outperforms the ERS model This finding highlights that code-related features contribute to modeling students' programming learning preferences.

- RNN-based sequential models exhibit superior capabilities in capturing learning behaviors. Our PERS and ERS, which extends recurrent neural networks, achieve the best performance. Additionally, the GRU4Rec and GRU4Recf models, designed based on recurrent neural networks, outperform all other sequential methods. This observation suggests that RNNs are particularly adept at capturing sequential programming behaviors.

5.3 RQ2: Ablation Study

We conduct an ablation study on PERS to understand the importance of the primary components. We obtain five variants: 1) **PERS-ep**, which removes the exercise position encoding; 2) **PERS-cr**, which removes the code representation; 3) **PERS-pa**, 4) **PERS-ps**, and 5) **PERS-us** are another three variants that remove $\mathbf{PA_t}$, $\mathbf{PS_t}$ and $\mathbf{US_t}$, respectively. Figure 3 displays the results of the PERS model and the its variants on the CodeNet-len and CodeNet-time datasets. From the figure, we can observe:



(a) CodeNet-len (b) CodeNet-time

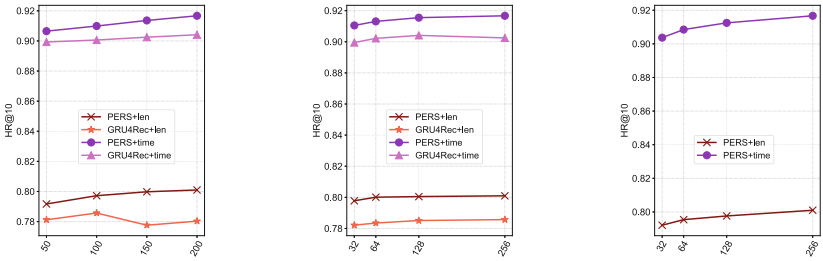
Fig. 3. Ablation Study Results on CodeNet-len(left) and CodeNet-time(right)

- Both the representating and updating modules play a crucial role in capturing programming behaviour. As can be observed, the removal of any component of the PERS adversely affects its performance, which emphasizes the rationality and effectiveness of the proposed methods.
- The impact of different components varies across different stages of learning. Specifically, in the CodeNet-len dataset, the performance is significantly affected when removing the position encoding of exercises (PERS-ep variant). On the other hand, in the CodeNet-time dataset, the performance sharply declines when the code representation (PERS-cr variant) is removed. This is because the CodeNet-len dataset comprises the latter part of students'

behavioral sequences, where students have developed a fixed behavioral pattern. Consequently, the representation of the exercises significantly impacts the model's performance. Similarly, removing different intrinsic latent vectors leads to different degrees of performance decline. The finding indicates that processing style is more critical in the initial learning stages while the understanding style are more influential as the learning pattern becomes more fixed.

5.4 RQ3: Sensitivity Analysis of Hyperparameters

We conduct a sensitivity analysis on the hyperparameter of PERS with two datasets: CodeNet-len and Code-time. In particular, we study three main hyperparameters: 1) sequence length $\lambda \in \{50, 100, 150, 200\}$, 2) dimension of exercise embedding $d_p \in \{32, 64, 128, 256\}$, and 3) dimension of code embedding $d_c \in \{32, 64, 128, 256\}$. In our experiments, we vary one parameter each time with others fixed. Figure 4 illustrates the impacts of each hyperparameter and we can obtain the following observations:



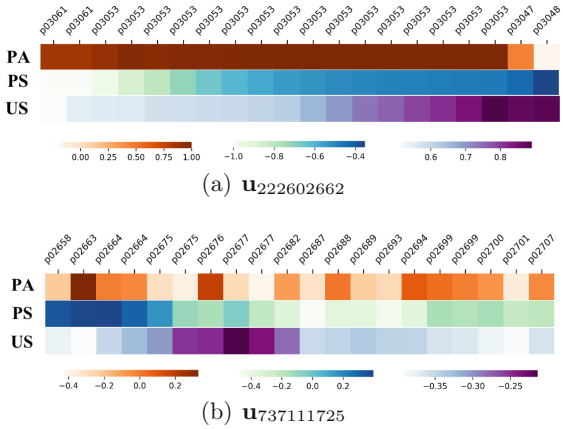
(a) Sequence Length λ (b) Exercise Embedding d_p (c) Code Embedding d_c

Fig. 4. Influence of three key hyperparameters on the performance of the PERS.

- Our model is capable of capturing long sequence dependencies. In Fig. 4(a), PERS performs better as the sequence length increases, while the results of GRU4Rec remain unchanged or even decline.
- As shown in Fig. 4(b), the performance of both PERS and GRU4Rec initially improves and then declines as the dimension of exercise embedding increases. The optimal performance is achieved at approximately $d_p = 128$.
- As the dimension of code embedding increases, the performance of PERS in Fig. 4(c) shows a consistent enhancement, highlighting the significance of code features in capturing programming learning patterns.

5.5 RQ4: Case Study on Visualization Analysis

To demonstrate the interpretability of our approach, we conduct a visualization analysis of three latent vectors involved in the PERS, i.e., programming ability



(a) $\mathbf{u}_{222602662}$

(b) $\mathbf{u}_{737111725}$

Fig. 5. Case study on latent vectors visualization

$\mathbf{PA}_t$, processing style $\mathbf{PA}_t$ and understanding style $\mathbf{US}_t$. We randomly selected the behavioral sequences of two students from the CodeNet dataset for the case study. From the exercise sequence of each student, we can observe that $\mathbf{u}_{222602662}$ tends to make multiple attempts at the same exercise and solve problems in a systematic manner, while $\mathbf{u}_{737111725}$ prefers solving problems by leaps and bound. We extract these three instinct vectors from the last time step of the model and visualize the dimensional reduction results in Figure 5. We note some observations in the visualization results:

- The extent of variation in students' programming abilities differs between inter-exercise and intra-exercise. Taking $\mathbf{u}_{222602662}$ as an example, as he made multiple attempts on $\mathbf{p}_{03053}$, his programming ability continuously improved. However, when he attempted the next exercise, his $\mathbf{PA}_t$ showed a noticeable decline. Therefore, fine-grained modeling of inter-exercise contributes to better capturing students' learning state.
- The changing patterns of learning styles among different students is consistent with their learning process. For $\mathbf{u}_{222602662}$, the value of $\mathbf{PS}_t$ and $\mathbf{US}_t$ gradually approach 1 during the programming learning process, suggesting a reflective and sequential learning style. As for $\mathbf{u}_{737111725}$, his corresponding latent vectors exhibit a gradual tendency towards -1, indicating an active and global learning style. This shows that the latent vectors can learn valuable information, thereby validating the rationality of our model.

6 Conclusions

In this paper, we study programming exercise recommendation (PER) to enhance engagement on online programming learning platforms. To solve PER, we propose a novel model called PERS based on simulating learners' intricate

programming behaviors. First, we extend the Felder-Silverman learning style model to the programming learning domain and present the programming learning style. After that, based on the programming learning style, we construct latent vectors to model learner's states, including programming ability, processing style, and understanding style. In particular, we introduce a differentiating module to update the states based on enhanced context, which are positions for exercises and compilation results for codes, respectively. Finally, the updated states at the last time step are sent to predict. Extensive experiments on two real-world datasets demonstrate the effectiveness and interpretability of our approach. In future work, we will explore incorporating the difference of structural features from students' submitted code to further enhance the performance of the model.

Acknowledgement. This work has been supported by the National Natural Science Foundation of China under Grant No. U1911203, and the National Natural Science Foundation of China under Grant No. 62377012.

References

1. Bagher, R.C., Hassanpour, H., Mashayekhi, H.: User trends modeling for a content-based recommender system. Expert Syst. Appl. **87**, 209–219 (2017)
2. Cai, C., He, R., McAuley, J.J.: SPMC: socially-aware personalized Markov chains for sparse sequential recommendation. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, 19–25 August 2017, pp. 1476–1482. ijcai.org (2017)
3. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4171–4186 (2019)
4. Felder, R.M., Silverman, L.K., et al.: Learning and teaching styles in engineering education. Eng. Educ. **78**(7), 674–681 (1988)
5. Felder, R.M., Spurlin, J.: Applications, reliability and validity of the index of learning styles. Int. J. Eng. Educ. **21**(1), 103–112 (2005)
6. Feng, Z., et al.: CodeBERT: a pre-trained model for programming and natural languages. In: Cohn, T., He, Y., Liu, Y. (eds.) Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16–20 November 2020. Findings of ACL, vol. EMNLP 2020, pp. 1536–1547. Association for Computational Linguistics (2020)
7. Gong, J., et al.: Reinforced MOOCs concept recommendation in heterogeneous information networks. CoRR abs/2203.11011 (2022)
8. Gong, J., Wang, S., Wang, J., Feng, W., Peng, H., Tang, J., Yu, P.S.: Attentional graph convolutional networks for knowledge concept recommendation in MOOCs in a heterogeneous view. In: Huang, J.X., Chang, Y., Cheng, X., Kamps, J., Murdock, V., Wen, J., Liu, Y. (eds.) Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, 25–30 July 2020, pp. 79–88. ACM (2020)
9. He, R., McAuley, J.J.: Fusing similarity models with Markov chains for sparse sequential recommendation. In: Bonchi, F., Domingo-Ferrer, J., Baeza-Yates, R.,

Zhou, Z., Wu, X. (eds.) IEEE 16th International Conference on Data Mining, ICDM 2016, 12–15 December 2016, Barcelona, Spain, pp. 191–200. IEEE Computer Society (2016)
10. Hidasi, B., Karatzoglou, A., Baltrunas, L., Tikk, D.: Session-based recommendations with recurrent neural networks. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, 2–4 May 2016, Conference Track Proceedings (2016)
11. Hidasi, B., Quadrana, M., Karatzoglou, A., Tikk, D.: Parallel recurrent neural network architectures for feature-rich session-based recommendations. In: Sen, S., Geyer, W., Freyne, J., Castells, P. (eds.) Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, 15–19 September 2016, pp. 241–248. ACM (2016)
12. Hou, Y., Hu, B., Zhang, Z., Zhao, W.X.: CORE: simple and effective session-based recommendation within consistent representation space. In: Amigó, E., Castells, P., Gonzalo, J., Carterette, B., Culpepper, J.S., Kazai, G. (eds.) SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, 11–15 July 2022, pp. 1796–1801. ACM (2022)
13. Huang, R., Lu, R.: Research on content-based MOOC recommender model. In: 5th International Conference on Systems and Informatics, ICSAI 2018, Nanjing, China, 10–12 November 2018, pp. 676–681. IEEE (2018)
14. Jiang, L., et al.: Eduhawkes: a neural Hawkes process approach for online study behavior modeling. In: Demeniconi, C., Davidson, I. (eds.) Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, 29 April–1 May 2021, pp. 567–575. SIAM (2021)
15. Jiang, L., Zhang, W., Wang, Y., Luo, N., Yue, L.: Augmenting personalized question recommendation with hierarchical information for online test platform. In: Li, B., et al. (eds.) ADMA 2022, Part I. Lecture Notes in Computer Science, vol. 13087, pp. 103–117. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-95405-5_8
16. Kang, W., McAuley, J.J.: Self-attentive sequential recommendation. In: IEEE International Conference on Data Mining, ICDM 2018, Singapore, 17–20 November 2018, pp. 197–206. IEEE Computer Society (2018)
17. Kolb, D.A.: Learning style inventory (1999)
18. Li, R., et al.: PST: measuring skill proficiency in programming exercise process via programming skill tracing. In: Amigó, E., Castells, P., Gonzalo, J., Carterette, B., Culpepper, J.S., Kazai, G. (eds.) SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, 11–15 July 2022, pp. 2601–2606. ACM (2022)
19. Ma, H., Huang, Z., Tang, W., Zhang, X.: Exercise recommendation based on cognitive diagnosis and neutrosophic set. In: 25th IEEE International Conference on Computer Supported Cooperative Work in Design, CSCWD 2022, Hangzhou, China, 4–6 May 2022, pp. 1467–1472. IEEE (2022)
20. Mao, K., et al.: Simplex: a simple and strong baseline for collaborative filtering. In: Demartini, G., Zuccon, G., Culpepper, J.S., Huang, Z., Tong, H. (eds.) CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, 1–5 November 2021, pp. 1243–1252. ACM (2021)
21. Muhammad, B.A., Qi, C., Wu, Z., Ahmad, H.K.: GRL-LS: a learning style detection in online education using graph representation learning. Expert Syst. Appl. **201**, 117138 (2022)

22. Pereira, F.D., et al.: Early dropout prediction for programming courses supported by online judges. In: Isotani, S., Millán, E., Ogan, A., Hastings, P., McLaren, B., Luckin, R. (eds.) AIED 2019. LNCS (LNAI), vol. 11626, pp. 67–72. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23207-8_13
23. Prithishkumar, I.J., Michael, S.A., et al.: Understanding your student: using the Vark model. J. Postgrad. Med. **60**(2), 183 (2014)
24. Puri, R., et al.: Codenet: a large-scale AI for code dataset for learning a diversity of coding tasks. In: Vanschoren, J., Yeung, S. (eds.) Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, Virtual (2021)
25. Ren, Y., Liang, K., Shang, Y., Zhang, Y.: Muloer-san: 2-layer multi-objective framework for exercise recommendation with self-attention networks. Knowl. Based Syst. **260**, 110117 (2023)
26. Shen, S., et al.: Learning process-consistent knowledge tracing. In: Zhu, F., Ooi, B.C., Miao, C. (eds.) KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, 14–18 August 2021, pp. 1452–1460. ACM (2021)
27. Sun, F., et al.: BERT4rec: sequential recommendation with bidirectional encoder representations from transformer. In: Zhu, W., et al. (eds.) Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, 3–7 November 2019, pp. 1441–1450. ACM (2019)
28. Tan, Q., et al.: Sparse-interest network for sequential recommendation. In: Lewin-Eytan, L., Carmel, D., Yom-Tov, E., Agichtein, E., Gabrilovich, E. (eds.) WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, 8–12 March 2021, pp. 598–606. ACM (2021)
29. Tan, Y.K., Xu, X., Liu, Y.: Improved recurrent neural networks for session-based recommendations. In: Karatzoglou, A., et al. (eds.) Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, 15 September 2016, pp. 17–22. ACM (2016)
30. Vaswani, A., et al.: Attention is all you need. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA, pp. 5998–6008 (2017)
31. Wu, J., et al.: Self-supervised graph learning for recommendation. In: Diaz, F., Shah, C., Suel, T., Castells, P., Jones, R., Sakai, T. (eds.) SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, 11–15 July 2021, pp. 726–735. ACM (2021)
32. Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., Tan, T.: Session-based recommendation with graph neural networks. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, 27 January–1 February 2019, pp. 346–353. AAAI Press (2019)
33. Zhu, R., et al.: Programming knowledge tracing: a comprehensive dataset and A new model. In: Candan, K.S., Dinh, T.N., Thai, M.T., Washio, T. (eds.) IEEE International Conference on Data Mining Workshops, ICDM 2022 - Workshops, Orlando, FL, USA, 28 November–1 December 2022, pp. 298–307. IEEE (2022)