Cross-Lingual Adversarial Domain Adaptation for Novice Programming

Ye Mao,[1] **Farzaneh Khoshnevisan,** [2] **Thomas Price,** [1] **Tiffany Barnes,** [1] **Min Chi** [1]

[1] North Carolina State University, {ymao4, twprice, tmbarnes, mchi}@ncsu.edu
[2] Intuit Inc., farzaneh_khoshnevisan@intuit.com

Abstract

Student modeling sits at the epicenter of adaptive learning technology. In contrast to the voluminous work on student modeling for well-defined domains such as algebra, there has been little research on *student modeling in programming (SMP)* due to data scarcity caused by the unbounded solution spaces of open-ended programming exercises. In this work, we focus on two essential SMP tasks: *program classification* and *early prediction of student success* and propose a *Cross-Lingual Adversarial Domain Adaptation (CrossLing)* framework that can leverage a large programming dataset to learn features that can improve SMP's build using a much smaller dataset in a different programming language. Our framework maintains one globally invariant latent representation across both datasets via an adversarial learning process, as well as allocating domain-specific models for each dataset to extract local latent representations that cannot and should not be united. By separating globally-shared representations from domain-specific representations, our framework outperforms existing state-of-the-art methods for both SMP tasks.

1 Introduction

Student modeling is a task of measuring students' performance in a learning environment and predicting their future performance based on their previous interactions with the environment. While student modeling has been extensively studied in well-defined domains like algebra and physics (Lin, Shen, and Chi 2016; Pardos and Heffernan 2010), student modeling for programming (SMP) remains an extremely challenging machine learning problem through the combination of *Knowledge Tracing* and *Code Representation*. Much of prior work has explored code representation (Mou et al. 2014; Allamanis, Peng, and Sutton 2016; Bui, Yu, and Jiang 2021), which is primarily concerned with identifying the functions the corresponding code belongs to. In SMP, however, it's vital to have the ability to measure students' performances in a learning environment and predict their future performance based on historical data (Geden et al. 2020). This former task is referred to as *program classification*: the task of determining whether a student's code is correct or not, while the latter is the task of *predicting a student's success at an early stage*. The latter task is particularly important for determining whether to provide additional learning interventions.

Programs are complex and contain both *structural/syntactic* and *semantic* information (Allamanis, Brockschmidt, and Khademi 2018). Traditionally, expert-designed features are used for SMP (Marwan et al. 2020) to represent student code heavily rely on expert knowledge of each task and are therefore labor-intensive and task-specific. Recently, deep learning-based models such as Code2vec (Alon et al. 2019) and ASTNN (Zhang et al. 2019) have demonstrated extraordinary results on public datasets by analyzing Abstract Syntax Trees (ASTs) of programs. These models, however, generally need a lot of data to perform well. For example, when it comes to program classification, ASTNN achieved 0.95 of AUC in CodeWorkout ($\mathcal{D}_2$) dataset with 795 submissions compared with 0.81 in iSnap ($\mathcal{D}_1$) dataset gathered from 171 students over four semesters. In real classroom settings, collecting programming sequences during small, in-person classes can be prohibitively expensive as most teachers don't teach using the same assignments in different semesters and it is rare to have a large dataset to cover all the possible solution space (Piech et al. 2015b). Domain Adaptation (DA), on the other hand, has emerged as a promising research direction for reducing human effort in data collection by learning sufficient information from relevant domains (Shen et al. 2018; Wang et al. 2019; Jiang et al. 2019). DA has been shown to enhance performance in a range of fields by sharing feature representations across different domains, wherein training data are obtained from multiple domains (Nam and Han 2016).

In this work, we propose a *Cross-lingual Adversarial Domain Adaptation* framework named **CrossLing** leveraging data from other programming systems to improve SMP in small classroom settings. Our proposed framework is based on a novel AST-based Neural Network (ASTNN) (see (Zhang et al. 2019) for more details), which has been shown to be very successful in student code analysis (Shi et al. 2021). CrossLing uses an adversarial learning process to separate global latent representations across all domains from domain-specific representations. In this context, domains refer to programming systems using different programming languages. We leverage student programming data from iSnap (domain $\mathcal{D}_1$) and CodeWorkout (do-

main $\mathcal{D}_2$). The effectiveness of CrossLing is evaluated on two tasks: student program classification and student success early predictions for $\mathcal{D}_1$. For the former, our results show that CrossLing indeed outperforms the state-of-the-art methods such as ASTNN and Code2Vec and other baselines. For the more challenging task of student success early prediction, we further combine CrossLing with temporal deep learning models such as Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) and Time-aware LSTM (T-LSTM) (Baytas et al. 2017), referred to as **L-CrossLing** and **TL-CrossLing** respectively. LSTM is designed to handle *sequences of events with discrete time steps* while T-LSTM can model the dynamics of student knowledge state *in continuous time* by taking the *time elapsed between successive elements (Time-Aware)* in a student's trajectory into consideration. As far back as the mid-1950s, student response time has been considered an indicator of student proficiency (Schnipke and Scrams 2002) as it demonstrates how active and accessible student knowledge is (Thomas et al. 1986). Our results show that TL-CrossLing performs significantly better than other student modeling methods, including those using the gold standard: expert-designed features.

Our main contributions can be summarized as follows:

- To the best of our knowledge, our proposed CrossLing framework is the first attempt to utilize adversarial domain adaptation in programming domains, especially across different programming languages.
- We extend CrossLing by integrating LSTM and T-LSTM to deal with the temporal nature in student programming sequences with/without time-irregularities.
- We explore the robustness and the effectiveness of our CrossLing framework and proposed the temporal models on two student modeling tasks.

2 Methods

Problem Description: Let's assume that we have 2 domains: $\mathcal{D}_1$ and $\mathcal{D}_2$. A domain $\mathcal{D}_i$ contains n_i student code snapshots represented as $\mathcal{D}_i = \{\mathbf{x}^k_{\mathcal{D}_i}, \mathrm{y}^k_{\mathcal{D}_i}\}_{k=1}^{n_i}$ where $\mathrm{y}^k \in \{0, 1\}$ is the outcome label and each student code snapshot $\mathbf{x}^k$ in $\mathcal{D}_i$ consists of a single submitted code snapshot x^k or a sequence of code snapshots: $\mathbf{x}^k = \{\mathrm{x}^k_1, ..., \mathrm{x}^k_{T_k}\}$, where x^k_t represents the student's code at time step t in $\mathbf{x}^k$ and T_K is the total number of steps taken by student k. In this work, for example, $\mathcal{D}_1$ is iSnap, which consists of time series of code snapshots from a relatively small group of students in a classroom setting, while $\mathcal{D}_2$ is CodeWorkout collected from 795 students, one submission per student. In our problem setting, we assume that $n_1 << n_2$ and our objective is to learn common knowledge from both domains to enhance predictions for domain $\mathcal{D}_1$. To do so, we apply adversarial learning to optimize the *global* representation for both domains as well as *locally* domain-specific knowledge.

For the task of program classification, the goal is to determine whether a code submission $\mathrm{x}^k_{T_k}$ by student k in $\mathcal{D}_1$ is correct or not. For the early prediction task, we aim to predict a student's future success based on partial sequence $\{\mathrm{x}^k_1, ..., \mathrm{x}^k_t\}$ where $t < T_k$ in the small domain $\mathcal{D}_1$. To do so, we leverage data from domain $\mathcal{D}_2$, with a much larger number of n_2 students working on similar tasks. In the following sections, we omit index k hereafter when it does not cause ambiguity for simplicity.

2.1 CrossLing

CrossLing is an ASTNN-based adversarial domain adaptation framework that learns both globally-shared latent representations and locally domain-specific information. Our key insight is that there are shared "global" features across programming languages and courses, which can be learned in one domain (with more data available) and applied in another. However, the challenge is isolating these "global" features from the "local" ones, which cannot and should not be applied across domains. Fig. 1 shows the architecture of the proposed CrossLing framework. Specifically, a local ASTNN is used to capture domain-specific skills/features. And the global ASTNN learns to address the cross-lingual knowledge shared among different programming languages. The proposed CrossLing framework exploits both local and global feature spaces of the domain representations. To accomplish this goal, we set different loss objectives. A source domain loss $\mathcal{L}_{\text{src}}$ is used to optimize the local classifier based on both local and global representations. A classifier loss $\mathcal{L}_{\text{clf}}$ is employed to optimize the global classifier based only on global representations. A difference loss $\mathcal{L}_{\text{diff}}$ is introduced to better splits global and local feature spaces. An adversarial loss $\mathcal{L}_{\text{adv}}$ prevents the algorithm from identifying the features' original domains in the shared space. The whole framework is trained and optimized by minimizing the following loss:

$$\mathcal{L} = \mathcal{L}_{\text{src}} + \alpha\mathcal{L}_{\text{clf}} + \beta\mathcal{L}_{\text{diff}} + \gamma\mathcal{L}_{\text{adv}} \quad (1)$$

where α, β, and γ are hyper-parameters used to weigh the importance of each loss term.

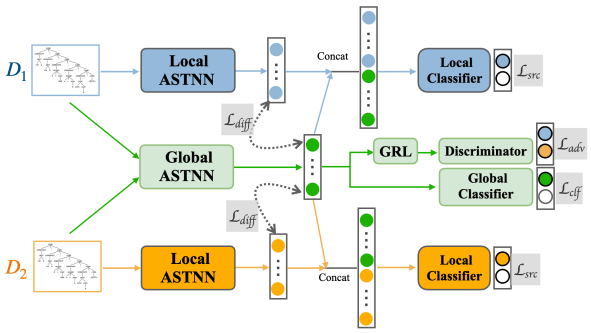


Figure 1: CrossLing model structure

CrossLing pre-trains local ASTNNs to generate local latent representations and ensures that the global latent representations are different from the local ones by maximizing a dissimilarity measure. A discriminator and a classifier based on global representations are employed to ensure domain-invariant and class-discriminative projection. In the following, we describe the two steps for training the CrossLing framework.

Step 1: Pre-train Local ASTNNs
The pre-training phase is comprised of two parts. In the first part, we pre-train the embedding matrix for both programming domains. In this work, we applied word2vec with skip gram to learn the embedding matrix for both block-based $\mathcal{D}_1$ and text-based $\mathcal{D}_2$ programming languages (Bui, Jiang, and Yu 2018). In the second part, we transform the raw code to appropriate input for ASTNN using the embedding matrix learned from the first part, and then train a standard ASTNN classifier for each domain separately (without global ASTNN part). The pre-trained ASTNNs will be loaded to initialize the local ASTNNs of CrossLing.

Step 2: Discriminative Adversarial Learning
As shown in Fig. 1, our framework is composed of two pre-trained local ASTNNs, one for each domain from step 1, and one global ASTNN that will generate global latent representations. The discriminator aligns the global representations from two domains, and the classifier learns to predict the learning outcome. Below we describe the loss functions for all components in the CrossLing framework, including the global and local ASTNNs, discriminator, and classifiers.

1. **Global and Local ASTNNs**: The network parameters of two local ASTNNs are initialized based on the pre-trained ASTNNs to generate local representations. In the global ASTNN, input data from both domains is utilized to generate their global representations. Therefore, the training dataset for the global ASTNN is the union of the two domains, with size $n_1 + n_2$. In our framework, the global classifier only consumes the global latent representations from the global ASTNN. By contrast, we join the cross-domain (global) and domain-specific (local) representations to build local classification models for each source domain.

   A simple fully connected neural network is used for classification, for all local and global classifiers. The network is optimized based on the binary cross-entropy loss in Eq. 2, where y is the label linked to each input program and $\hat{y}$ is the output from each classifier, and $\mathbf{\Theta}$ represents the relevant network parameters.

$$\mathcal{L}(\hat{y}, y; \mathbf{\Theta}) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y})) \quad (2)$$

   Therefore, we have:

$$\begin{aligned} \mathcal{L}_{\text{src}} &= w_1 \mathcal{L}(\hat{y}_{\mathcal{D}_1}, y_{\mathcal{D}_1}; \mathbf{\Theta_{\mathcal{D}_1}}) + w_2 \mathcal{L}(\hat{y}_{\mathcal{D}_2}, y_{\mathcal{D}_2}; \mathbf{\Theta_{\mathcal{D}_2}}) \\ \mathcal{L}_{\text{clf}} &= \mathcal{L}(\hat{y}_g, y_g; \mathbf{\Theta_g}) \end{aligned} \quad (3)$$

   where w_1 and w_2 are the weights for each source domain. Stochastic gradient descent is used to update all the parameters based on back-propagation calculation.

2. **Difference Loss**: To encourage the separation of local and global feature representations, we add a dissimilarity measure defined by a Frobenius norm (Khoshnevisan and Chi 2020), which measures the orthogonality between global and local representations, and zero indicates orthogonal vectors. Let us denote matrices $\boldsymbol{Z}^g_{\mathcal{D}_1}$ and $\boldsymbol{Z}^g_{\mathcal{D}_2}$ as hidden global matrices. Similarly, $\boldsymbol{Z}^l_{\mathcal{D}_1}$ and $\boldsymbol{Z}^l_{\mathcal{D}_2}$ indicate hidden local representations. Therefore, the difference loss is defined as:

$$\mathcal{L}_{\text{diff}} = \left\| \mathbf{Z}^g_{\mathcal{D}_1}{}^\top \mathbf{Z}^l_{\mathcal{D}_1} \right\|_F^2 + \left\| \mathbf{Z}^g_{\mathcal{D}_2}{}^\top \mathbf{Z}^l_{\mathcal{D}_2} \right\|_F^2 \quad (4)$$

   where $\|\cdot\|_F^2$ refers to the squared Frobenius norm.

3. **Discriminator**: The shared feature space of the global ASTNN stores the common knowledge learned from both source domains, which is used to improve the classification performance. Thus, the shared feature space contains global features, and not domain-related information. That is, its predictions should be made based on global features that cannot discriminate between different source domains. We use a Gradient Reversal Layer (GRL) (Ganin et al. 2016) to achieve minimax optimization, and a domain classifier trained to predict the domain producing the hidden representations.

   During adversarial training, given a code snippet, the discriminator $\boldsymbol{\theta}_{\text{d}}$ is optimized to determine the code's source domain, while the global ASTNN $\boldsymbol{\theta}_{\text{g}}$ is trained to confuse it by learning features that represent common knowledge across the two domains. In this paper, the adversarial loss $\mathcal{L}_{\text{adv}}$ is defined as:

$$\begin{aligned} \mathcal{L}_{\text{adv}} = \min_{\boldsymbol{\theta}_{\text{d}}} \max_{\boldsymbol{\theta}_{\text{g}}} (& \sum_{j \in [\mathcal{D}_1, \mathcal{D}_2]} \sum_{i \in n_j} d_i^j \log(\hat{d}_i^j) \\ & + (1 - d_i^j) \log(1 - \hat{d}_i^j)) \end{aligned} \quad (5)$$

   where n_j is the size of input data in domain j. During training, we maximize cross entropy for domain classification with respect to the discriminator $\boldsymbol{\theta}_{\text{d}}$, meanwhile minimizing it with respect to the global ASTNN $\boldsymbol{\theta}_{\text{g}}$.

2.2 Temporal and Time-aware Frameworks

We combine CrossLing with temporal models to handle sequences of student programming data (code). In prior research (Mao et al. 2020), T-LSTM was shown to deliver better performance by modeling the dynamics of student knowledge *in continuous time*, than standard LSTM using *discrete timesteps*. To investigate the relative performance of LSTM and T-LSTM on our task, we implement both models within the CrossLing framework, denoted as **L-CrossLing** and **TL-CrossLing** respectively. We contrast these models with another proposed model, **TL-ASTNN**. The following sections will provide detailed descriptions of these models.

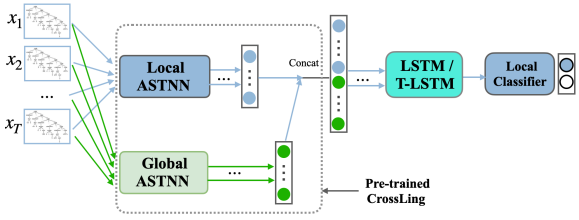


Figure 2: L-CrossLing and TL-CrossLing model structure: L-CrossLing uses LSTM for temporal modeling and TL-CrossLing utilizes T-LSTM

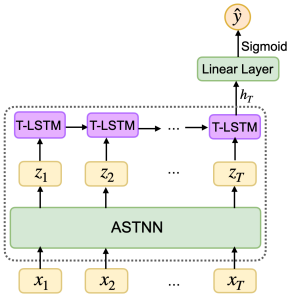


Figure 3: TL-ASTNN model structure

L-CrossLing and TL-CrossLing The architecture of L-CrossLing and TL-CrossLing is shown in Fig. 2. The main difference between the two is that L-CrossLing uses LSTM to learn the temporal information while TL-CrossLing applies T-LSTM. To be specific, the temporal CrossLing framework contains two main parts: 1) local and global ASTNNs for code representation pre-trained from CrossLing; and 2) LSTM/T-LSTM to handle the temporal information. More specifically, in **Step 1: Pre-train CrossLing**, we first pre-train the original CrossLing framework as described earlier in Sec. 2.1. Then we load the weights of local ($\mathcal{D}_1$) and global ASTNNs into our temporal CrossLing framework (L-CrossLing or TL-CrossLing). And then in **Step 2: Optimize Training**, input data are fed into both local and global ASTNNs to generate corresponding representations and then passed to the connected LSTM or T-LSTM layer for evaluation. And we utilize the last-time output from LSTM/T-LSTM to make the final predictions. The whole model will be optimized by minimizing the cross entropy loss given in Eq. 2.

$$\begin{aligned} &l_1, ..., l_T = \text{Local ASTNN}(x_1, ..., x_T) \\ &g_1, ..., g_T = \text{Global ASTNN}(x_1, ..., x_T) \\ &h_T = \text{LSTM / T-LSTM}([l_1, g_1], ..., [l_T, g_T]) \\ &\hat{y} = \text{sigmoid}(\mathbf{W}h_T + b) \end{aligned} \tag{6}$$

TL-ASTNN As shown in Fig. 3, TL-ASTNN can be interpreted as an extension of Temporal-ASTNN (Mao et al. 2021) (denoted as **L-ASTNN** (L for LSTM), to handle the time irregularity in student data. More specifically, the only difference is that in L-ASTNN, the output of ASTNN Z_t, is used as an input for the connected LSTM cell; while in TL-ASTNN, we pass both Z_t and the corresponding time interval Δt to the T-LSTM cell to model the temporal dynamics of student programming skills. With time as an input, TL-ASTNN is more time-aware so we expect it to make better predictions than L-ASTNN. The training process for TL-ASTNN can be summarized as follows:

$$\begin{aligned} &z_1, ..., z_T = \text{ASTNN}(x_1, ..., x_T) \\ &h_T = \text{T-LSTM}(z_1, ..., z_T) \\ &\hat{y} = \text{sigmoid}(\mathbf{W_1}h_T + b_l) \end{aligned} \tag{7}$$

where $\mathbf{W_1}$ is the weight matrix and b_l is the bias term for the connected linear layer. Still, the whole framework will be optimized by minimizing the cross entropy loss (Eq. 2).

3 Datasets[1]

3.1 iSnap

iSnap ($\mathcal{D}_1$), is an extension to Snap*!* (Garcia, Harvey, and Barnes 2015), a block-based programming environment, used in an introductory computing course for non-majors in North Carolina State University (Price, Dong, and Lipovac 2017). iSnap maintains a code trace log which records all student programming actions/steps (e.g. adding or deleting a block), a snapshot of the resulting code, and the *time* taken for each step. In prior research, an expert feature detector has been proposed to automatically detect 7 expert features of a student snapshot (Zhi et al. 2018). Those *expert-designed* features are binary and indicate whether the corresponding feature presents in student code or not. In this work, we focus on one homework exercise named Squiral, derived from the BJC curriculum (Garcia, Harvey, and Barnes 2015). In Squiral, students are asked to write a procedure that draws a square-like spiral; correct solutions use at least 7 lines of code with procedures, loops, and variables. In general, students spend about 20 minutes and use 10 to 2000 steps to complete it. We collected Squiral data from 4 semesters in years 2016 and 2017, with 65, 38, 29, and 39 student code traces respectively (see Appendix for detail).

3.2 CodeWorkout

CodeWorkout ($\mathcal{D}_2$) [2], is an open online system for programming in Java. It provides a web-based platform on which students from various backgrounds can practice programming and instructors can offer courses (Edwards and Murali 2017). When students "submit" code, the system provides detailed pass/fail feedback on its expert-designed test cases. In this work, we focus on one programming exercise named isEverywhere, where the knowledge of loops and arrays are evaluated. As in prior research (Shi et al. 2021), we use the first submission from each student. This resulted in 795 student code submissions, which represent code snapshots as in iSnap, but only at the time when students first submit their code for feedback. Different from iSnap, CodeWorkout does *not* log student code traces during programming, but only their submissions. As a result, only submissions from students are recorded and sequences of student edits are not available.

4 Task 1: Student Program Classification

In the task of student program classification, we aim to predict the correctness of students' submissions (i.e. correct or incorrect) in iSnap ($\mathcal{D}_1$). The effectiveness of CrossLing is compared against standard ASTNN and other baselines.

[1] All iSnap data were obtained anonymously through an exempt IRB-approved protocol, and we scored them using binary rubrics. All Code Workout data were shared under Creative Commons license, and were de-indentified and automatically scored against test cases. No demographic data or grades were collected. This research seeks to remove societal harms that come from lower engagement and retention of students who need more personalized interventions during programming.

[2] https://codeworkout.cs.vt.edu/

4.1 Experiments

We evaluate the effectiveness of our proposed CrossLing in terms of its ability to accurately classify student code in *iSnap* ($\mathcal{D}_1$); and more importantly, if it improves the performance of original ASTNN. Thus, we train CrossLing with data from both iSnap and CodeWorkout, but only test it on *iSnap*. In contrast, the training and testing sets for other baselines contain only iSnap data.

Single Domain Baselines We explore two types of baseline models, including three token-based classic machine learning (ML) models and two AST-based deep learning (DL) models. The classic ML models include K-Nearest Neighbors (**KNN**), Logistic Regression (**LR**), and Support-Vector Machine (**SVM**). Following prior token-based approaches, we applied TF-IDF to extract textual features (Zhang et al. 2019) via pre-order traversal of the original ASTs. The DL group contains **Code2Vec** and **ASTNN**, both of which have been shown to be very successful in source code analysis by using the structural information from ASTs. To be specific, Code2Vec takes a set of leaf-to-leaf AST paths as input, and the inputs for ASTNN are a set of statement-level subtrees. For all five models in this group, only iSnap data was used for training and testing.

CrossLing For the domain adaptation group, our proposed CrossLing framework is explored. During training, we take the hyperparameters that achieve the best performance on the development set via a small grid search over combinations of the batch size $\in \{16, 32, 64\}$, learning rate $\in$ [0.001, 0.1], $\alpha, \beta, \gamma \in$ [0.01, 1].

It is important to emphasize that all models are evaluated using semester-based temporal cross-validation (3-fold) in this task, which only applied data from previous semesters for training and is a much stricter approach than the standard cross-validation (Mao et al. 2020). We report Accuracy, Precision, Recall, F1 Score, and AUC (Area Under ROC curve). Accuracy and AUC are considered the most important metrics in the educational technology domain; and AUC is believed to be generally more robust.

4.2 Results

Table 1: Student Program Classification Results in iSnap

Model	Accuracy	Precision	Recall	F1	AUC
Majority Baseline	0.6321	-	-	-	0.5000
KNN	0.6132	0.7321	0.6119	0.6667	0.6137
LR	0.6604	0.8298	0.5821	0.6842	0.6885
SVM	0.6604	0.7460	0.7015	0.7231	0.6456
Code2Vec	0.6810	0.8038	0.6786	0.7239	0.7017
ASTNN	0.8113	**0.8730****	0.8209	0.8462	0.8079
CrossLing	**0.8491****	0.8592	**0.9104****	**0.8841****	**0.8271****

Note: best model in **bold** and **

Table 1 compares the performance of the seven models for domain $\mathcal{D}_1$, iSnap. Overall, our results show that CrossLing is the best among all the models for the task of student program classification. Specifically, CrossLing achieves around 4% and 2% improvement on Accuracy and AUC over standard ASTNN, respectively. These results suggest that by learning the common knowledge from different domains, our proposed domain adaptation approach can help to improve the performance of ASTNN and also address the lack of sufficient labeled data. Given its effectiveness, we further explore the temporal CrossLing models on the task of student success early prediction.

5 Task 2: Student Success Early Prediction

In the second task, we are given the first *up to* n *minutes* of a student's sequence data and our goal is to predict whether the student will successfully complete the programming assignment within one hour (Mao et al. 2020). It is worth noting that student success early prediction is a much more challenging task compared to program classification: 1) it involves temporal information; and 2) the observation windows are very early (up to first 10 minutes) and thus students' final submissions are not available for training or testing (Mao et al. 2021).

5.1 Experiments

To further explore the power of our CrossLing framework, we compare it with two other *code representation* groups: expert-designed features (Expert) and standard ASTNN on the early prediction task. For each group (Expert, ASTNN, CrossLing), we explore three different models: the "last value"-based (Batal et al. 2012) Logistic Regression (**LR**) models, the *temporal* **LSTM** models, and the *time-aware* LSTM (**T-LSTM**) models. Here, *last-value* models cannot handle sequential data, *temporal* models are able to do it but don't consider time-irregularity, while *time-aware* models take the elapsed time between consecutive steps into consideration and are therefore aware of time. For example, when our observation window is the first 2 minutes, we apply the last value before 2 minutes for each sequence and treat them as inputs for the last-value model, while all the sequences that occur within first 2 minutes will be used for LSTM and T-LSTM; meanwhile, T-LSTM takes another sequences of time intervals as input.

Three Expert Models The three models in the Expert group are trained on expert-designed features for the iSnap dataset ($\mathcal{D}_1$). Specifically, **Expert** applies last-value LR; **L-Expert** is based on LSTM and **TL-Expert** utilizes T-LSTM.

Three ASTNN Models There are three ASTNN-based models including last-value **ASTNN**, temporal **L-ASTNN** and our proposed **TL-ASTNN**, all of them use raw student iSnap code from $\mathcal{D}_1$ as input.

Three CrossLing Models We have **CrossLing**, **L-CrossLing**, and **TL-CrossLing** in this group. Different from the former two groups, these models are trained on data from both iSnap ($\mathcal{D}_1$) and CodeWorkout ($\mathcal{D}_2$).

Excepting that LR is implemented using the sklearn[3] library in python, all the other models are implemented in Pytorch[4] using a mini-batch stochastic optimizer with grid search of batch sizes in $\{16, 32, 64\}$. The same experimental

[3]https://scikit-learn.org/

[4]https://pytorch.org/

setup is used for all models with 100 epochs and early stopping. The embedding size for all ASTNNs/CrossLings is 128 and LSTM/T-LSTM hidden size is set to 64. All the sequences are zero-padded to have the same length. More importantly, all experimental results are reported for semester-based cross-validation and we consider Accuracy and AUC as the most important metrics as in the first task (Sec. 4).

5.2 Results

We present our results of student success early prediction in the following two parts: 1) we first compare the effectiveness of all nine models on first-2-minute early prediction; and 2) we explore their average performance across different observation windows varying from 2 to 10 minutes with every 2-minute interval (2, 4, 6, 8, 10).

Table 2: Student Success Early Predictions at First-2-minute

Model	Accuracy	Precision	Recall	F1	AUC
Majority Baseline	0.6604	-	-	-	0.5000
Expert Group					
Expert	0.6509	**0.7260**	0.7571	0.7413	**0.6008**
L-Expert	0.6698	0.6699	**0.9857****	0.7977	0.5206
TL-Expert	**0.6886**	0.6869	0.9714‡	**0.8047**	0.5552
ASTNN Group					
ASTNN	0.6604	0.7024	0.8429	0.7662	0.5742
L-ASTNN	0.7170	0.7381	**0.8857**	0.8052	0.6373
TL-ASTNN	**0.7358**	**0.7561**	**0.8857**	**0.8158**	**0.6651**
CrossLing Group					
CrossLing	0.7075	0.7079	**0.9429**	0.8098	0.5964
L-CrossLing	0.7736‡	0.7875 ‡	0.9000	**0.8400**	0.7139 ‡
TL-CrossLing	**0.7830****	**0.8310****	0.8429	0.8396‡	**0.7548****

Note: best models in each group are in **bold**
the overall best and second best are labeled with ** and ‡, respectively.

Results at First-2-minute Table 2 shows different performance measures of all the nine models at first-2-minute, together with the majority baseline. In terms of Accuracy, neither the Expert nor the ASTNN model beat the simple majority baseline (0.6604). This is not surprising, as relying only on the last snapshot in the first-2-minute is too early for these models to make effective early predictions. The fact that across the three last-value based models (Expert, ASTNN, CrossLing), the best performance comes from our proposed CrossLing model suggests that learning common knowledge from domain adaptation does help us to understand student code better. Also, all *temporal* models achieve better performance than their corresponding *last-value* models. This is reasonable since *temporal* models are able to capture the temporal information related to student success from the sequences, but such information is not available to *last-value* models. Between the two temporal models (LSTM and T-LSTM), we can see that T-LSTM models consistently outperform LSTM models within the same group. In other words, incorporating time-awareness might assist us in understanding student learning progression.

Overall, TL-CrossLing achieves the best performance in the first 2-minute observation window, especially with a 9% improvement on AUC over TL-ASTNN and a 15% improvement over using Expert alone. This suggests that by integrating domain adaptation and time-awareness into TL-CrossLing, it can gain both structural and temporal knowledge from student code.

Results in First-10-minute Overall The comparison of all models for the early prediction of student success in first 10 minutes is reported in Table 3. Here we report the mean value and corresponding standard deviation (in parenthesis) for each evaluation metric. Following prior research (Kurmi, Kumar, and Namboodiri 2019), we also report the Critical Difference (CD) diagram for Nemenyi test (Wilcoxon Signed Ranks Tests showed the same results) in Fig. 4.

Table 3: Average Performance of Student Success Early Predictions in first 10 minutes

Model	Accuracy	Precision	Recall	F-measure	AUC
Majority Baseline	0.6604	-	-	-	0.5000
Expert Group					
Expert	0.6566 (±0.05)	0.8209 ‡ (±0.05)	0.6229 (±0.06)	0.7017 (±0.06)	0.6725 (±0.05)
L-Expert	0.7170 (±0.03)	0.7299 (±0.04)	**0.9314**** (±0.03)	**0.8168** (±0.02)	0.6240 (±0.06)
TL-Expert	**0.7453** (±0.04)	0.7927 (±0.06)	0.8457 (±0.07)	0.8135 (±0.03)	**0.6979** (±0.06)
ASTNN Group					
ASTNN	0.6698 (±0.02)	0.7463 (±0.03)	0.7686 (±0.01)	0.7517 (±0.03)	0.6232 (±0.03)
L-ASTNN	0.7396 (±0.02)	0.7597 (±0.03)	0.8914‡ (±0.04)	0.8190 (±0.01)	0.6679 (±0.04)
TL-ASTNN	**0.7698** (±0.03)	**0.7961** (±0.02)	0.8771 (±0.04)	**0.8341** (±0.02)	**0.7191** (±0.03)
CrossLing Group					
CrossLing	0.7151 (±0.02)	0.7459 (±0.03)	0.8686 (±0.05)	0.8008 (±0.01)	0.6426 (±0.04)
L-CrossLing	0.7793‡ (±0.01)	0.7987 (±0.02)	0.8914‡ (±0.02)	0.8421‡ (±0.01)	0.7263‡ (±0.02)
TL-CrossLing	**0.7899**** (±0.01)	**0.8471**** (±0.02)	0.8657 (±0.04)	**0.8556**** (±0.01)	**0.7856**** (±0.02)

Note: best models in each group are in **bold**
the overall best and second best are labeled with ** and ‡, respectively.

Table 3 shows a similar pattern as we observed earlier in Table 2. In each group (Expert, ASTNN, CrossLing), T-LSTM models outperform LSTMs and LRs. Furthermore, when implementing the same temporal (or last-value) models, CrossLing is generally shown to be the best one. Overall, TL-CrossLing is the best model and it achieves the best Accuracy, Precision, F1 and AUC (a 7% improvement over TL-ASTNN) in first 10 minutes, followed by L-CrossLing with the second-best Accuracy, Recall, F1 and AUC. Therefore, our temporal CrossLing models are the best two models for student success early predictions in first 10 minutes.

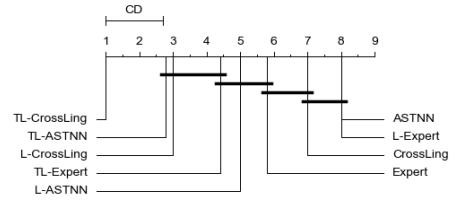
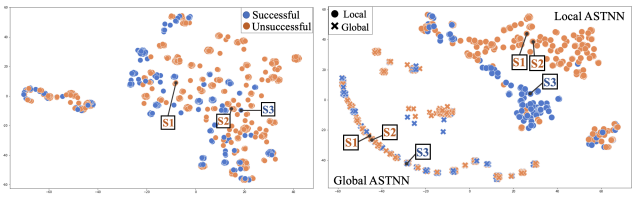


Figure 4: CD diagram: unconnected models mean pairwise significance (e.g. proposed TL-CrossLing is significantly different from all other models), confidence level is 0.05.

Feature Visualization To illustrate the difference in performance between ASTNN and CrossLing models, we visualize the location of three student code snapshots: $S1$ and $S2$ are two unsuccessful code snapshots while $S3$ is a successful one. Fig. 5 shows the t-SNE visualization (Van der Maaten and Hinton 2008) of the learned feature representations for both models on the student success prediction task. In the figure, blue and orange markers represent successful and unsuccessful iSnap ($\mathcal{D}_1$) samples in the first 2 minutes, respec-



(a) Standard ASTNN (b) CrossLing

Figure 5: Visualization for student code in iSnap ($\mathcal{D}_1$)

tively. • and × in Fig. 5(b) are local and global representations, respectively. As we can see in Fig. 5(a), $S1$ is far from $S2$ in the ASTNN model's feature space, while $S3$ is very close to $S2$. It seems that the ASTNN model fails to differentiate $S3$ from $S2$, and misses the similarities between $S1$ and $S2$ that make them unsuccessful (e.g. failing to update a key variable). In contrast, Fig. 5(b) shows that the CrossLing feature representations for $S1$ and $S2$ are similar, and both of them are well-separated from $S3$ across both local and global feature spaces. Moreover, the successful and unsuccessful groups are better clustered in the CrossLing model's local representations when the domain-invariant knowledge is isolated and learnt by the global feature space. That is, our proposed CrossLing model picks up the information lost by standard ASTNN through learning the cross-domain and domain-specific knowledge together.

6 Related Work

Student Modeling Student modeling has been widely and extensively explored by utilizing student sequences. Prior research has proposed a series of approaches mostly for well-defined domains, such as Item Response Theory (IRT) (Tatsuoka 1983), Performance Factor Analysis (PFA) (Pavlik, Cen, and Koedinger 2009), Bayesian Knowledge Tracing (BKT) (Corbett and Anderson 1994). Recently, deep learning models, especially Recurrent Neural Network (RNN) or RNN-based models such as LSTM have also been explored in student modeling and have shown superior performance (Piech et al. 2015a; Tang, Peterson, and Pardos 2016; Khajah, Lindsey, and Mozer 2016; Xiong et al. 2016).

Programming, by contrast, has been relatively underexplored for student modeling. Wang et al. (2017) applied a recursive neural network similar to (Piech et al. 2015b) as the embedding for student submission sequence, then fed them into a 3-layer LSTM to predict the student's future performance. On the other hand, Emerson et al. (2019) have utilized four categories of features: prior performance, hint usage, activity progress, and interface interaction to evaluate the accuracy of Logistic Regression models for multiple block-based programming activities. More recently, Dong et al. (2021) proposes a data-driven method that uses student trace logs to identify struggling moments during a programming assignment and determine the appropriate time for an intervention. As far as we know, our proposed temporal CrossLing framework is the first attempt to address student modeling for novice programming through adversarial domain adaptation across different programming languages.

Domain Adaptation Numerous approaches have previously been proposed to address adaptation needs that arise in different application scenarios, such as image recognition, multi-language text classification, sentiment classification, human activity classification, etc. (Ling et al. 2008; Zhuang et al. 2019; Khoshnevisan and Chi 2020). The main challenge of cross-domain learning is how to reduce the discrepancies in data distributions across domains. One line of research is focusing on adversarial learning, which is designed to minimise the approximate discrepancy distance between different domains. For example, domain adversarial neural network (DANN) employs a gradient reversal layer (GRL) and learns domain-invariant features by a minimax game between the domain classifier and the feature extractor (Ganin and Lempitsky 2014). Adversarial discriminative domain adaptation (ADDA) uses GAN (Goodfellow et al. 2014) with general loss in a non-shared weight architecture (Tzeng et al. 2017). Our proposed CrossLing framework is also an adversarial adaptation method as it learns and evaluates global representations in an adversarial manner.

Within the field of NLP, deep learning methods form another line of work to automatically produce superior feature representations for cross-domain scenarios (Ganin et al. 2016; Liu, Qiu, and Huang 2017; Joty et al. 2017). These deep networks explore effective domain discrepancy measurement and matching methods to boost performance (Glorot, Bordes, and Bengio 2011; LeCun, Bengio, and Hinton 2015). For example, the multi-domain adaption adversarial network (MDANet) is proposed to alleviate the domain discrepancy based on explicitly learning a shared feature space across different domains (Ding et al. 2019). This architecture is similar to ours, while we are focusing on a more challenging programming scenario involving different programming systems with different programming languages.

7 Conclusions

Developing a robust, generalizable model for the early prediction of student learning progression is a crucial yet challenging task. With jobs in computing projected to grow 13% from 2020 to 2030, the need for *automated support for learning programming* is growing. Research shows that student modeling can improve learning by 1-2 standard deviations when done well by adapting difficulty and interventions to students' needs (Arroyo et al. 2007). Educational environments, with SMP, stand to benefit tens of millions of students learning programming in K-12 and CS classes. In this work, we demonstrate the effectiveness of CrossLing on both student program classification and student success early predictions. Empirically, we show that CrossLing-based models outperform state-of-the-art methods because of their ability to separate local information from global representations and leverage the common knowledge from different domains. In future work, we plan to investigate our framework on other tasks or different domains to explore whether it consistently supports improvement for programming environments.

8 Acknowledgements

This research was supported by the NSF Grants: Integrated Data-driven Technologies for Individualized Instruction in STEM Learning Environments (1726550), EXP: Data-Driven Support for Novice Programmers (1623470), CAREER: Improving Adaptive Decision Making in Interactive Learning Environments (1651909), and Generalizing Data-Driven Technologies to Improve Individualized STEM Instruction by Intelligent Tutors (2013502).

References

Allamanis, M.; Brockschmidt, M.; and Khademi, M. 2018. Learning to Represent Programs with Graphs. In *International Conference on Learning Representations*.

Allamanis, M.; Peng, H.; and Sutton, C. 2016. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*, 2091–2100. PMLR.

Alon, U.; Zilberstein, M.; Levy, O.; and Yahav, E. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL): 1–29.

Arroyo, I.; Ferguson, K.; Johns, J.; Dragon, T.; Meheranian, H.; Fisher, D.; Barto, A.; Mahadevan, S.; and Woolf, B. P. 2007. Repairing disengagement with non-invasive interventions. In *AIED*, volume 2007, 195–202.

Batal, I.; Fradkin, D.; Harrison, J.; Moerchen, F.; and Hauskrecht, M. 2012. Mining recent temporal patterns for event detection in multivariate time series data. In *SIGKDD*, 280–288. ACM.

Baytas, I. M.; Xiao, C.; Zhang, X.; Wang, F.; Jain, A. K.; and Zhou, J. 2017. Paient subtyping via time-aware LSTM networks. In *SIGKDD*. ACM.

Bui, N. D.; Jiang, L.; and Yu, Y. 2018. Cross-language learning for program classification using bilateral tree-based convolutional neural networks. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Bui, N. D.; Yu, Y.; and Jiang, L. 2021. TreeCaps: Tree-Based Capsule Networks for Source Code Processing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 30–38.

Corbett, A. T.; and Anderson, J. R. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *UMUAI*, 4(4): 253–278.

Ding, X.; Shi, Q.; Cai, B.; Liu, T.; Zhao, Y.; and Ye, Q. 2019. Learning multi-domain adversarial neural networks for text classification. *IEEE Access*, 7: 40323–40332.

Dong, Y.; Marwan, S.; Shabrina, P.; Price, T.; and Barnes, T. 2021. Using Student Trace Logs To Determine Meaningful Progress and Struggle During Programming Problem Solving. In *Proceedings of the 14th International Conference on Educational Data Mining*.

Edwards, S. H.; and Murali, K. P. 2017. CodeWorkout: short programming exercises with built-in data collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, 188–193.

Emerson, A.; Rodríguez, F. J.; Mott, B.; Smith, A.; Min, W.; Boyer, K. E.; Smith, C.; Wiebe, E.; and Lester, J. 2019. Predicting Early and Often: Predictive Student Modeling for Block-Based Programming Environments. *International Educational Data Mining Society*.

Ganin, Y.; and Lempitsky, V. 2014. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*.

Ganin, Y.; Ustinova, E.; Ajakan, H.; Germain, P.; Larochelle, H.; Laviolette, F.; Marchand, M.; and Lempitsky, V. 2016. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1): 2096–2030.

Garcia, D.; Harvey, B.; and Barnes, T. 2015. The Beauty and Joy of Computing. *ACM Inroads*, 6(4): 71–79.

Geden, M.; Emerson, A.; Rowe, J.; Azevedo, R.; and Lester, J. 2020. Predictive student modeling in educational games with multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 654–661.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.

Jiang, H.; Wang, H.; Hu, W.; Kakde, D.; and Chaudhuri, A. 2019. Fast incremental SVDD learning algorithm with the Gaussian kernel. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 3991–3998.

Joty, S.; Nakov, P.; Màrquez, L.; and Jaradat, I. 2017. Cross-language learning with adversarial neural networks: Application to community question answering. *arXiv preprint arXiv:1706.06749*.

Khajah, M.; Lindsey, R. V.; and Mozer, M. C. 2016. How deep is knowledge tracing? *arXiv preprint arXiv:1604.02416*.

Khoshnevisan, F.; and Chi, M. 2020. An adversarial domain separation framework for septic shock early prediction across ehr systems. In *2020 IEEE International Conference on Big Data (Big Data)*, 64–73. IEEE.

Kurmi, V. K.; Kumar, S.; and Namboodiri, V. P. 2019. Attending to discriminative certainty for domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 491–500.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature*, 521(7553): 436–444.

Lin, C.; Shen, S.; and Chi, M. 2016. Incorporating Student Response Time and Tutor Instructional Interventions into Student Modeling. In *UMAP*, 157–161. ACM.

Ling, X.; Xue, G.-R.; Dai, W.; Jiang, Y.; Yang, Q.; and Yu, Y. 2008. Can chinese web pages be classified with english data source? In *Proceedings of the 17th international conference on World Wide Web*, 969–978.

Liu, P.; Qiu, X.; and Huang, X. 2017. Adversarial multi-task learning for text classification. *arXiv preprint arXiv:1704.05742*.

Mao, Y.; Marwan, S.; Price, T. W.; Barnes, T.; and Chi, M. 2020. What Time is It? Student Modeling Needs to Know. In *Proceedings of The 13th International Conference on Educational Data Mining (EDM 2020)*, 171–182.

Mao, Y.; Shi, Y.; Marwan, S.; Price, T. W.; Barnes, T.; and Chi, M. 2021. Knowing When and Where: Temporal-ASTNN for Student Learning Progression in Novice Programming Tasks. In *Proceedings of The 14th International Conference on Educational Data Mining (EDM 2021)*.

Marwan, S.; Gao, G.; Fisk, S.; Price, T. W.; and Barnes, T. 2020. Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, 194–203.

Mou, L.; Li, G.; Jin, Z.; Zhang, L.; and Wang, T. 2014. TBCNN: A tree-based convolutional neural network for programming language processing. *arXiv preprint arXiv:1409.5718*.

Nam, H.; and Han, B. 2016. Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4293–4302.

Pardos, Z. A.; and Heffernan, N. T. 2010. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *UMAP*, 255–266. Springer.

Pavlik, P. I.; Cen, H.; and Koedinger, K. R. 2009. Performance Factors Analysis –A New Alternative to Knowledge Tracing. In *AIED*, 531–538. ISBN 978-1-60750-028-5.

Piech, C.; Bassen, J.; Huang, J.; Ganguli, S.; Sahami, M.; Guibas, L. J.; and Sohl-Dickstein, J. 2015a. Deep knowledge tracing. In *NIPS*, 505–513.

Piech, C.; Huang, J.; Nguyen, A.; Phulsuksombati, M.; Sahami, M.; and Guibas, L. 2015b. Learning program embeddings to propagate feedback on student code. In *International conference on machine Learning*, 1093–1102. PMLR.

Price, T. W.; Dong, Y.; and Lipovac, D. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, 483–488. ISBN 9781450346986.

Schnipke, D. L.; and Scrams, D. J. 2002. Exploring issues of examinee behavior: Insights gained from response-time analyses. *Computer-based testing: Building the foundation for future assessments*, 237–266.

Shen, J.; Qu, Y.; Zhang, W.; and Yu, Y. 2018. Wasserstein distance guided representation learning for domain adaptation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Shi, Y.; Mao, T.; Barnes, T.; Chi, M.; and Price, T. 2021. More with less: Exploring how to use deep learning effectively through semi-supervised learning for automatic bug detection in student code. In *In Proceedings of the 14th International Conference on Educational Data Mining (EDM) 2021*.

Tang, S.; Peterson, J. C.; and Pardos, Z. A. 2016. Deep neural networks and how they apply to sequential education data. In *L@S*, 321–324. ACM.

Tatsuoka, K. 1983. Rule space: An approach for dealing with misconceptions based on item response theory. *JEM*, 20(4): 345–354.

Thomas, R. D. L. V. S.; et al. 1986. *Response Times: Their Role in Inferring Elementary Mental Organization: Their Role in Inferring Elementary Mental Organization*. Oxford University Press, USA.

Tzeng, E.; Hoffman, J.; Saenko, K.; and Darrell, T. 2017. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7167–7176.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

Wang, L.; Sy, A.; Liu, L.; and Piech, C. 2017. Learning to Represent Student Knowledge on Programming Exercises Using Deep Learning. *International Educational Data Mining Society*.

Wang, X.; Li, L.; Ye, W.; Long, M.; and Wang, J. 2019. Transferable attention for domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 5345–5352.

Xiong, X.; Zhao, S.; Van Inwegen, E.; and Beck, J. 2016. Going Deeper with Deep Knowledge Tracing. In *EDM*, 545–550.

Zhang, J.; Wang, X.; Zhang, H.; Sun, H.; Wang, K.; and Liu, X. 2019. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 783–794. IEEE.

Zhi, R.; Price, T. W.; Lytle, N.; Dong, Y.; and Barnes, T. 2018. Reducing the State Space of Programming Problems through Data-Driven Feature Detection. In *EDM (Workshops)*.

Zhuang, F.; Qi, Z.; Duan, K.; Xi, D.; Zhu, Y.; Zhu, H.; Xiong, H.; and He, Q. 2019. A Comprehensive Survey on Transfer Learning. *arXiv preprint arXiv:1911.02685*.