Code Semantics Learning with Deep Neural Networks: an AI-based Approach for Programming Education

Md. Mostafizer Rahman[1,2][0000−0001−9368−7638], Yutaka Watanobe[2][0000−0002−0030−3859], Paweł Szmeja[3][0000−0003−0869−3836], Piotr Sowiński[3,4][0000−0002−2543−9461], Marcin Paprzycki[3][0000−0002−8069−2152], and Maria Ganzha[4][0000−0001−7714−4844]

[1] Dhaka University of Engineering & Technology, Gazipur, Bangladesh
`mostafiz26@gmail.com`, `mostafiz@duet.ac.bd`
[2] The University of Aizu, Japan
`yutaka@u-aizu.ac.jp`
[3] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
`firstname.lastname@ibspan.waw.pl`
[4] Warsaw University of Technology, Warsaw, Poland

Abstract. Modern programming languages are very complex, diverse, and non-uniform in their structure, code composition, and syntax. Therefore, it is a difficult task for computer science students to retrieve relevant code snippets from large code repositories, according to their programming course requirements. To solve this problem, an AI-based approach is proposed, for students to better understand and learn code semantics, with solutions for real-world coding exercises. First, a large number of solutions are collected from a course titled "Algorithms and Data Structures" and preprocessed, by removing unnecessary elements. Second, the solution code is converted into a sequence of words and tokenized. Third, the sequence of tokens is used to train and validate the model, through a word embedding layer. Finally, the model is used for the relevant code retrieval and classification task, for the students. In this study, a bidirectional long short-term memory neural network (BiLSTM) is used as the core deep neural network model. For the experiment, approximately 120,000 real-world solutions from three datasets are used. The trained model achieved an average precision, recall, F1 score, and accuracy of 94.35%, 94.71%, 94.45%, and 95.97% for the code classification task, respectively. These results show that the proposed approach has potential for use in programming education.

Keywords: Code Semantics · Deep Learning · Programming Education · Software Engineering · Natural Language Processing

1 Introduction

Nowadays, the importance of programming is increasing due to the information and communication technology (ICT) needs of modern society. Thus, higher educational institutions are placing more emphasis on improving students' programming skills [15, 13]. In addition, there is a relationship between programming and other academic courses. For instance, a data-driven analysis found that better programming skills have a positive impact on overall academic performance [15]. Therefore, it is important to provide more meaningful and effective support for the programming learning process. In this case, an AI model can provide more dynamic support, such as code searching, relevant code suggestions, algorithm identification, and classification. In particular, code classification is considered one of the main foundations of many of coding-related activities. However, classifying a large number of code fragments [5], in a non-automated manner, is a formidable task. Considering this problem, the aim of this work is to establish if machine learning models can be effectively used to understand the meaning of code and become the basis for supporting coding-related activities.

In recent years, many academic and industrial research experiments have been conducted to facilitate programming tasks. For example, identifying the location of errors in code [18, 5], error detection, code refactoring [16, 10], code evaluation and repair [9], etc. In addition, deep neural network (DNN) models are effectively used for coding tasks [2]. Considering the need to handle lines of programming code, recurrent neural networks (RNNs) are widely used to develop models for programming tasks. RNN models include long short-term memory (LSTM) and bi-directional long short-term memory (BiLSTM), and their variants are also commonly used in this context. In [21], structural features of the solution code were used to identify the algorithm in the code, using the convolutional neural network (CNNs) model. Also, binary code classification, code completion, code repair, and code evaluation are performed using different BiLSTM and LSTM models [9, 8].

Due to its unidirectional processing of input sequences [3], the LSTM model performed worse than the BiLSTM [19] model in coding tasks [14]. On the other hand, the BiLSTM model can process input sequences in both directions (forward and backward). Since variables, functions, and classes may depend on past or future lines of code, the BiLSTM model is more effective in such cases. The aim of this contribution is to build an AI engine, using BiLSTM neural networks, to better understand the semantics of code, and to generate better code-related services. For this purpose, real-world program code fragments have been collected from an "Algorithms and Data Structures" (ADS) programming course. Next, data preprocessing, word tokenization, word embedding, model training, validation, and evaluation were performed. The experimental results show that the model achieves significant success in code classification. Moreover, the trained model can be integrated with an AI-enabled web client, in a programming learning platform, to provide students more with convenient coding-related services. Hence, this work delivers the following key contributions:

– An AI engine based on BiLSTM neural networks was trained on a large dataset of real-world solution code collected from a programming course. The

[5] The terms code, source code, solution code, and program code are used to denote a similar meaning.

engine was used to build an AI-based framework for learning code semantics. Experimental results show that the trained model obtained very good results in classifying code fragments.
- The integration scheme of the proposed approach with a programming learning platform is investigated.
- The proposed code semantics learning model can be useful for teachers to identify students' deficiencies in programming, and these problems can be discussed in class.

2 Related Work

Programming techniques and programming environments have evolved significantly over the past few decades, and intelligent coding platforms such as Crescendo [20], scratch [17], and CloudCoder [4] have been used to alleviate selected basic challenges faced by programmers while coding. However, these platforms have had limited success when programmers attempted to solve highly structured problems, using C, C++, and Python [20]. On the other hand, programming support platforms are also being developed for learning programming, with many of them focused on only one programming language [24, 12, 25]. Abe et al. proposed a programming environment dedicated to the C language, with basic functions such as variable declarations, expressions, and statements [1]. Similarly, Nguyen and Chua [11] proposed a web-based interface for PHP programming. In their study, rules for various expressions, such as for-loop, while-loop, and equality conditions, are developed. Our proposed approach for learning code semantics using the BiLSTM model is different from the existing methods because the proposed AI model is designed to understand the semantics of multilingual (e.g., C, C++, and Python) code. Therefore, the proposed AI model can be more effective within a programming learning platform, to assist students in coding. Moreover, the proposed AI model was trained with diverse, real-world multilingual solution code fragments, in order to provide support for many programming languages.

3 Motivation

In a recent study [15], a data-driven analysis was conducted, based on evaluation logs of solution code in a programming learning platform. The evaluation logs were collected from the works of 357 students in the ADS course. The results showed that about 37.27% of all submissions were accepted, while the remaining 62.73% were rated as incorrect. It was also found that the error verdicts were categorized into 5 main groups, i.e., Wrong Answer (WA), Compile Error (CE), Presentation Error (PE), Runtime Error (RE), and Resource Limit Error (RLE). RLE includes Time Limit Exceedance (TLE), Memory Limit Exceedance (MLE), and Output Limit Exceedance (OLE). Figure 1 shows the detailed breakdown of errors. Here, approximately 44% were WA, 21.25% were CE, 13% were RE, 10.5% were PE, and 11.25% were RLE.

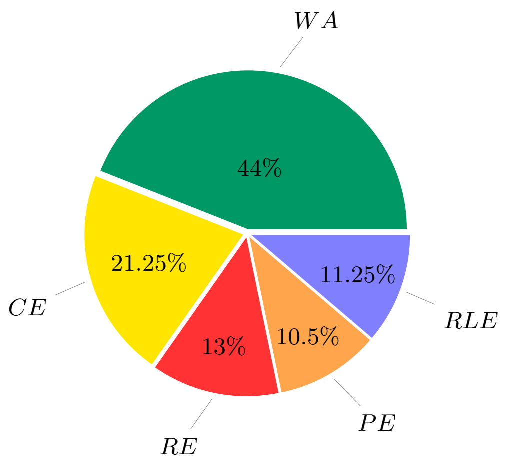


Fig. 1: Breakdown of errors, based on solution code evaluation.

In general, CE errors were caused by syntax errors in the solution code, while non-CE errors were caused by logical errors and inefficiency in the code. Figure 1 shows that a large percentage of submitted solutions received error decisions for the above reasons. To address this important problem in programming learning, and to help students improve their programming skills by providing coding-related services (e.g., code refactoring, code suggestions, code search, and classification), a machine learning model has been developed.

4 Proposed Approach

Figure 2 presents the outline of the proposed framework for AI-based code semantics learning. In the first stage of the process, the solution code fragments of the ADS course are collected from the Aizu Online Judge (AOJ) platform [22, 23]. AOJ is a globally-recognized learning and competition platform for programming. In the second stage, the code is preprocessed by removing irrelevant elements such as comments, whitespaces, and tabs[6].

Next, in the tokenization step, the preprocessed code fragments are converted into a sequence of words, and each word is encoded with a unique integer number. Let $I = \{i_1, i_2, i_3, \cdots, i_l\}$ be the sequence of words of a code fragment, and $K = \{k_1, k_2, k_3, \cdots, k_l\}$ be the corresponding integer IDs for words. The overall process of code tokenization is shown in Figure 3.

Each tokenized word sequence is converted into a vector of real numbers through the embedding layer. The embedding matrix is $\boldsymbol{S} \in \mathbb{R}^{l \times d}$, where l is the token dictionary size and d is the embedding size. For this study, the dimensions of the embedding matrix are $\boldsymbol{S} \in \mathbb{R}^{10000 \times 200}$. The vectorized data is passed

[6] Which elements are irrelevant depends on the programming language.

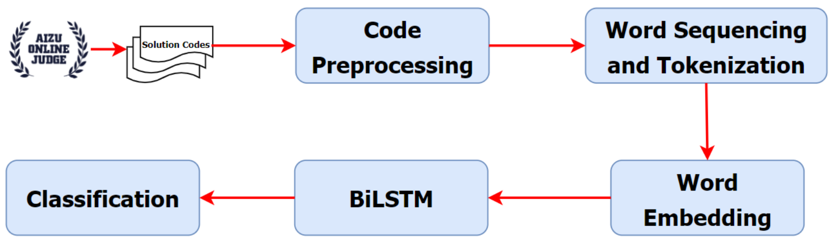


Fig. 2: AI-based code semantics learning framework

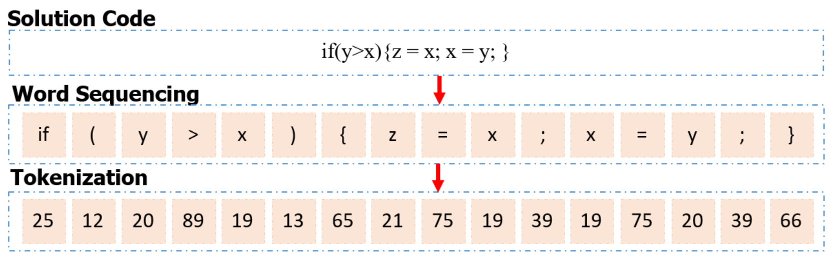


Fig. 3: Code tokenization process for the BiLSTM model training

to the BiLSTM neural network [19]. The bidirectional information processing nature of the BiLSTM model enables it to understand the complex context of the long dependent information, i.e., solution code fragments. Here, the final code classification is performed in the output layer.

Figure 4 presents the integration of the proposed AI Engine built with BiLSTM, the trained AI model, and the web client in the programming learning platform. To provide the programming services, the web client connects the AI Engine and the trained AI model.

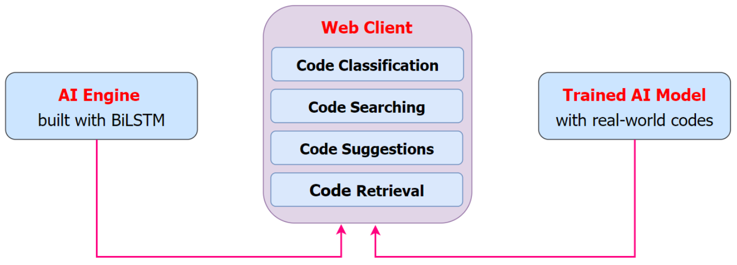


Fig. 4: AI Engine-enabled platform for programming learning

Table 1: Description of the datasets

Dataset	**Classes**	**Class names**	**Avg. sol. len.**	**# solutions**
Sorting	8	CountingSort, StableSort, BubbleSort, InsertionSort, MergeSort, SelectionSort, ShellSort, QuickSort	840.44	53,308
Searching	5	ExhaustiveSearch, StringSearch, LinearSearch, BinarySearch, PatternSearch	650.22	25,994
Graphs and trees	14	TreeWalk, BinaryTrees, MST (Minimum Spanning Tree), Projection, Graph, Area, BST (Binary Search Tree), Reflection, SP (Shortest Path), CBT (Complete Binary Tree), Convex, Puzzle, RootedTrees, Intersection	1658.46	38,761

5 Experimental Results

5.1 Datasets

For the experiments, all correct solutions from the AOJ platform, of the ADS course[7] were collected. There are 15 topics in this course and each topic involves solving 3 or 4 programming problems. The problems include various algorithms (e.g., sorting, searching, graphs, and trees), elementary data structures (e.g., stack, queue, and linked list), and numerical computations. For the experiments, three datasets were built, that group the problems thematically into: sorting, searching, and graphs and trees. Table 1 shows the details of the datasets including number of classes, average solution code length, and total number of solutions in each dataset.

5.2 Evaluation Methods

In this paper, we used four evaluation metrics, i.e., precision ($\mathcal{P}$), recall ($\mathcal{R}$), F1 score, and accuracy ($\mathcal{A}$), to evaluate the classification performance of the model. These methods are dependent on the false positive (fp), true positive (tp), false negative (fn), and true negative (tn) rates. A detailed description of these performance evaluation methods can be found in [14].

5.3 Implementation Details

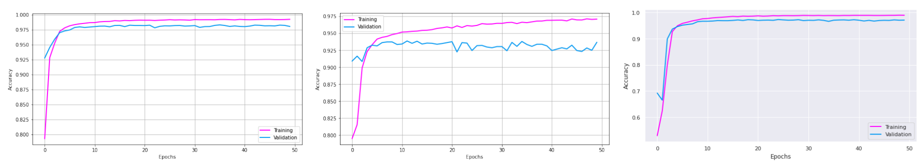
When implementing the BiLSTM classification model, several different sets of hyperparameters were tried, by manual fine-tuning, to achieve better results. In this paper, the reported number of training epochs is 50, the batch size is 32, the maximum code sequence length is 500, the learning rate is $\alpha =$

[7] https://onlinejudge.u-aizu.ac.jp/courses/lesson/1/ALDS1/1

$\{0.01, 0.005, 0.001\}$, the number of BiLSTM nodes is 400 ($\overrightarrow{200} + \overleftarrow{200}$), the number of nodes in the dense layer is 200, the training data ratio is 80%, the validation data ratio is 13%, the test data ratio is 7%, the activation function for the dense layer is ReLU [6] and the optimization function is Adam [7]. In addition, sparse categorical cross-entropy is used as the loss function ($\mathcal{L}$) for the multiclass classification model [19]. These hyperparameters resulted in best overall performance. However, the aim of this work was *not* to exhaustively search the hyperparameter space. Hence, no claim is made that this set of hyperparameters is optimal. All experimental tasks are computed in Google Colab using the Keras+TensorFlow (2.11.x) framework and Python 3.

5.4 Results

Figure 5 shows the accuracy of BiLSTM model training and validation, on all three datasets (e.g., sorting, searching, graphs and trees) when $\alpha = 0.001$. Figure 6, on the other hand, shows the model training and validation losses for these datasets. From Figures 5 and 6, the following observations can be made: (i) the model achieved the highest training and validation accuracy for the sorting dataset and the lowest for the graph and tree dataset, (ii) the validation accuracy curve in the searching dataset shows the inconsistency during validation, and (iii) the model has the highest validation losses for the searching dataset.

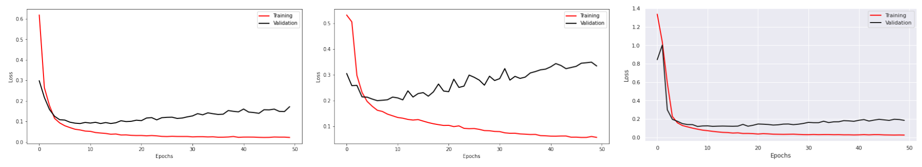


(a) Sorting dataset (b) Searching dataset (c) Graph and tree dataset

Fig. 5: Comparison of training and validation accuracy of BiLSTM models on three datasets

Tables 2, 3, and 4 show the class-wise $\mathcal{P}$, $\mathcal{R}$, and $F1$ values for three datasets. The following observations can be made from these tables: (i) the model achieved higher $\mathcal{P}$, $\mathcal{R}$, and $F1$ values between 96% and 99% for most classes in the sorting dataset, (ii) the model achieved relatively lower $\mathcal{P}$, $\mathcal{R}$, and $F1$ values for the *ExhaustiveSearch*, *BinarySearch*, and *LinearSearch* classes in the searching dataset, (iii) in contrast, the model obtained very low $\mathcal{P}$, $\mathcal{R}$, and $F1$ values for many classes in the graph and tree dataset. This is due to the greater heterogeneity, complexity, classes, and code length in the graph and tree dataset.

Table 5 shows the average $\mathcal{P}$, $\mathcal{R}$, and $F1$ values for all datasets. Experimental results show that the BiLSTM model achieved $\mathcal{P}$, $\mathcal{R}$, and $F1$ of 98.37%, 98.04%, and 98.20%, respectively, for the sorting dataset. This is due to the lower diversity



(a) Sorting dataset (b) Searching dataset (c) Graph and tree dataset

Fig. 6: Comparison of training and validation losses of BiLSTM models on three datasets

Table 2: Class-wise $\mathcal{P}$, $\mathcal{R}$, and $F1$ for the sorting dataset

Classes	$\boldsymbol{\mathcal{P}}$	$\boldsymbol{\mathcal{R}}$	$\boldsymbol{F1}$
CountingSort	0.98	0.96	0.97
StableSort	0.99	0.98	0.98
BubbleSort	0.97	0.99	0.98
InsertionSort	1.00	0.99	0.99
MergeSort	0.98	0.99	0.99
SelectionSort	0.97	0.96	0.96
ShellSort	0.99	0.99	0.99
QuickSort	0.99	0.98	0.99

Table 3: Class-wise $\mathcal{P}$, $\mathcal{R}$, and $F1$ for the searching dataset

Classes	$\boldsymbol{\mathcal{P}}$	$\boldsymbol{\mathcal{R}}$	$\boldsymbol{F1}$
ExhaustiveSearch	0.91	0.91	0.91
StringSearch	0.99	0.99	0.99
LinearSearch	0.90	0.91	0.90
BinarySearch	0.86	0.92	0.89
PatternSearch	0.99	0.99	0.99

Table 4: Class-wise $\mathcal{P}$, $\mathcal{R}$, and $F1$ for graph and tree dataset

Classes	$\boldsymbol{\mathcal{P}}$	$\boldsymbol{\mathcal{R}}$	$\boldsymbol{F1}$
TreeWalk	0.77	0.82	0.80
BinaryTrees	0.96	0.99	0.97
MST	0.98	0.98	0.98
Projection	0.99	0.99	0.99
Graph	0.93	0.95	0.94
Area	0.99	0.98	0.98
BST	0.89	0.83	0.86
Reflection	0.99	0.94	0.96
SP	0.87	0.71	0.78
CBT	1.00	0.99	1.00
Convex	0.50	0.71	0.59
Puzzle	0.98	0.98	0.98
RootedTrees	0.98	0.99	0.98
Intersection	1.00	0.98	0.99

and higher similarity of the solutions for the sorting algorithms. In contrast, the model achieved relatively low $\mathcal{P}$, $\mathcal{R}$, and $F1$ of 91.56%, 91.77%, and 91.45% for the graph and tree datasets, respectively.

Table 5: Average $\mathcal{P}$, $\mathcal{R}$, and $F1$ values for all datasets

Dataset	$\boldsymbol{\mathcal{P}}$**(%)**	$\boldsymbol{\mathcal{R}}$**(%)**	$\boldsymbol{F1}$**(%)**
Sorting	98.37	98.04	98.20
Searching	93.13	94.32	93.70
Graph and Tree	91.56	91.77	91.45

A comparison of $F1$ and $\mathcal{A}$ values is shown in Figure 7. It can be seen that (i) the $F1$ and $\mathcal{A}$ values for the datasets sorting and searching are mostly similar, and (ii) the difference between $F1$ and $\mathcal{A}$ value for the dataset graph and tree is comparatively high. This is due to the unbalanced instances in the classes of this dataset.

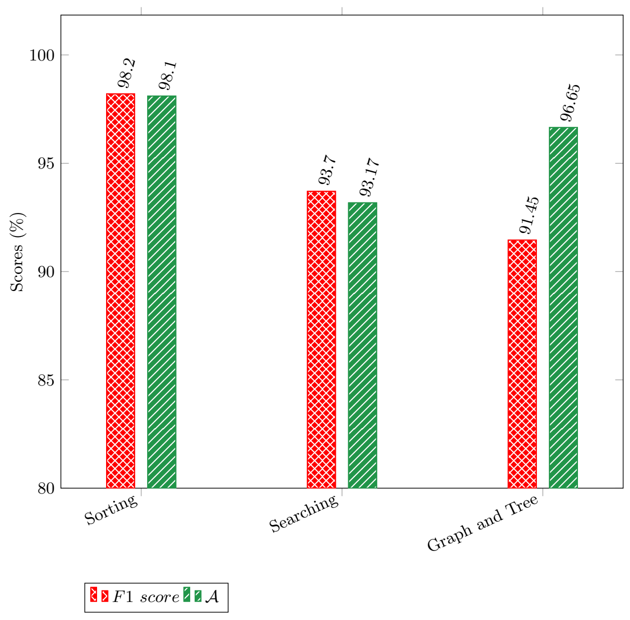


Fig. 7: Comparison of $F1$ and $\mathcal{A}$ values for all datasets

Separately, Figures 8, 9, and 10 show the confusion matrices for these three datasets, which also demonstrate the effectiveness of the BiLSTM model for classification tasks. These higher values of $F1$ and $\mathcal{A}$ ensure that the model better understands the semantics of the code.

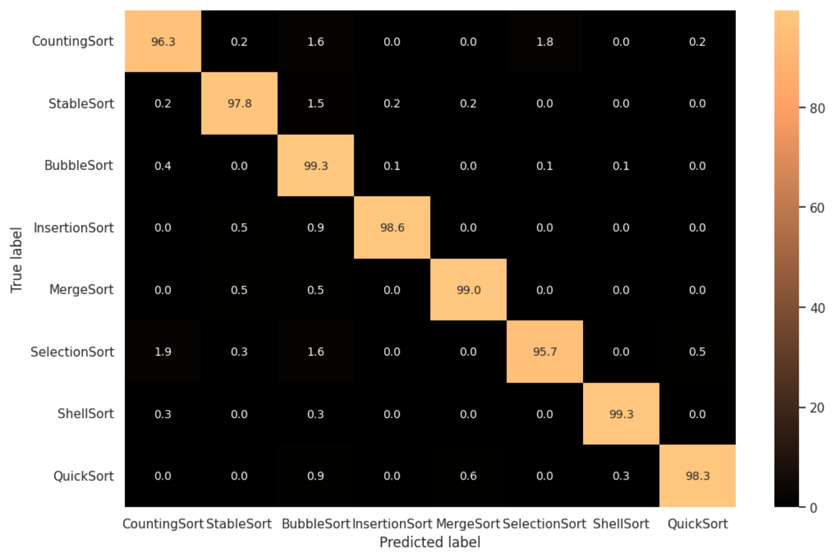


Fig. 8: Confusion matrix for Sorting dataset

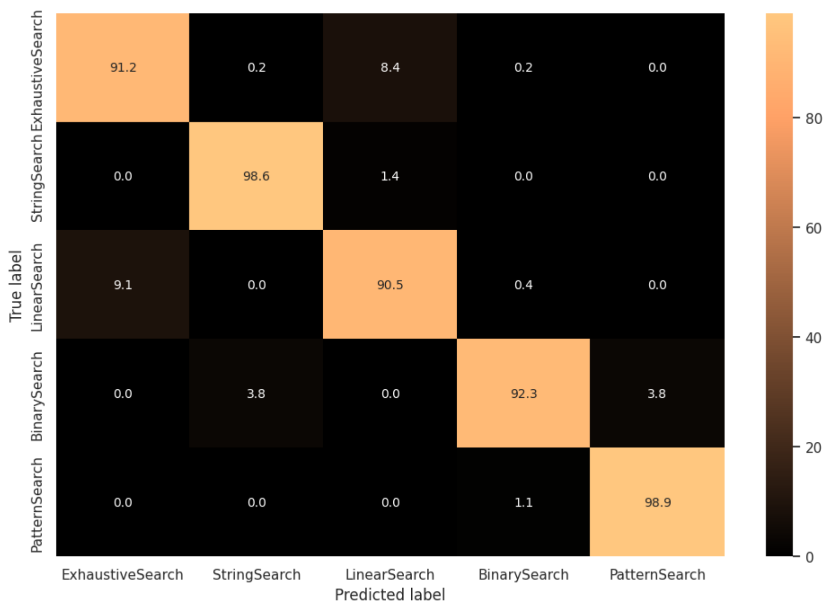


Fig. 9: Confusion matrix for Searching dataset

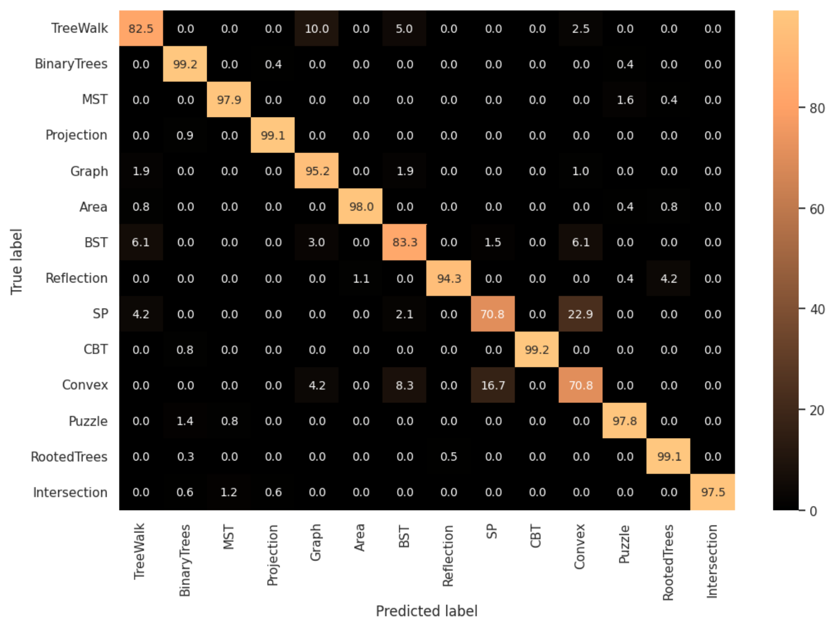


Fig. 10: Confusion matrix for Graph and Tree dataset

In addition, further experiments have been conducted with different learning rates i.e. $\alpha = \{0.01, 0.005, 0.001\}$, as shown in Table 6. It can be seen that the BiLSTM model achieves higher $\mathcal{P}$, $\mathcal{R}$, $F1$ and $\mathcal{A}$ values when $\alpha = 0.001$ is used for all datasets. It also indicates that, as could be expected, the model learns the semantics of the code better, when the learning rate becomes slower.

6 Discussion

A data-driven analysis shows that a significant number of solutions, provided by students, are evaluated as incorrect, i.e., 62.73% in a programming learning platform [15]. We believe that AI-based support can be useful for students in solving problems, and can reduce the number of wrong answers. In this paper, we investigated the performance of the BiLSTM neural network model for code semantics learning. For this purpose, real-world solution code fragments were collected, the code was preprocessed and tokenized, and the model was trained. In performed experiments, the model achieved an average $F1$ of 94.45% and $\mathcal{A}$ of 95.97% for all three datasets (see Table 5 and Figure 7). These results indicate that the BiLSTM model learned the semantics of the code well, which can be useful for coding-related services in the programming learning platform. More importantly, AI Engine enabled programming learning platform can be useful for students to understand the code errors, algorithms, and syntax, by

Table 6: Average $\mathcal{P}$, $\mathcal{R}$, $F1$, and $\mathcal{A}$ values for all datasets when $\alpha = \{0.01, 0.005, 0.001\}$

Dataset	$\boldsymbol{\alpha}$	$\boldsymbol{\mathcal{P}}$(%)	$\boldsymbol{\mathcal{R}}$(%)	$\boldsymbol{F1}$(%)	$\boldsymbol{\mathcal{A}}$
Sorting	0.010	96.49	96.09	96.26	95.79
	0.005	97.77	97.32	97.53	97.26
	0.001	98.37	98.04	98.20	98.10
Searching	0.010	92.54	91.98	92.14	92.08
	0.005	94.51	94.19	94.33	92.19
	0.001	93.13	94.32	93.70	93.17
Graph and Tree	0.010	85.14	83.83	83.86	93.99
	0.005	85.40	86.58	85.72	95.68
	0.001	91.56	91.77	91.45	96.65

receiving code suggestions, refactoring, and classification services during coding. Additionally, the proposed approach can support multilingual coding tasks.

7 Concluding remarks

In this study, code semantics learning approach using BiLSTM neural networks has been proposed. A substantial number of real-world solutions, delivered by the students of the University of Aizu, have been collected from an operational programming learning platform (i.e., AOJ). To this data, standard preprocessing was applied and obtained data was used to train and evaluate the BiLSTM model for three different datasets (e.g., sorting, searching, and graph and tree), while applying various hyperparameters. The model achieved an average $\mathcal{P}$, $\mathcal{R}$, $F1$, and $\mathcal{A}$ values of 94.35%, 94.71%, 94.45%, and 95.97%, respectively, in classifying solutions. These results also indicate that the BiLSTM model understands code semantics with a high degree of accuracy. Moreover, the proposed AI model can be useful in the programming learning platform to provide students with various programming-related services such as code suggestion, code refactoring, classification, etc. Future work will consider improving the model performance with more diverse datasets.

Acknowledgment

This research was financially supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI (Grant Number 19K12252).

References

1. Abe, K., Fukawa, Y., Tanaka, T.: Prototype of visual programming environment for c language novice programmer. In: 2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI). pp. 140–145 (2019). https://doi.org/10.1109/IIAI-AAI.2019.00037

2. Dam, K.H., Tran, T., Pham, T.: A deep language model for software code. ArXiv **abs/1608.02715** (2016)
3. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (11 1997). https://doi.org/10.1162/neco.1997.9.8.1735, https://doi.org/10.1162/neco.1997.9.8.1735
4. Hovemeyer, D., Spacco, J.: Cloudcoder: A web-based programming exercise system. J. Comput. Sci. Coll. **28**(3), 30 (jan 2013)
5. Huo, X., Li, M., Zhou, Z.H.: Learning unified features from natural and programming languages for locating buggy source code. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. p. 1606–1612. IJCAI'16, AAAI Press (2016)
6. Javid, A.M., Das, S., Skoglund, M., Chatterjee, S.: A relu dense layer to improve the performance of neural networks. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 2810–2814. IEEE (2021)
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980
8. M. Mostafizer, R., Watanobe, Y., Nakamura, K.: A neural network based intelligent support model for program code completion. Scientific Programming **2020** (2020). https://doi.org/10.1155/2020/7426461, https://doi.org/10.1155/2020/7426461
9. M. Mostafizer, R., Watanobe, Y., Nakamura, K.: A bidirectional lstm language model for code evaluation and repair. Symmetry **13**(2) (2021). https://doi.org/10.3390/sym13020247, https://doi.org/10.3390/sym13020247
10. Meng, N., Hua, L., Kim, M., McKinley, K.S.: Does automated refactoring obviate systematic editing? In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1, pp. 392–402 (2015). https://doi.org/10.1109/ICSE.2015.58
11. Nguyen, T., Chua, C.: A logical error detector for novice php programmers. In: 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 215–216 (2014). https://doi.org/10.1109/VLHCC.2014.6883062
12. Price, T.W., Dong, Y., Lipovac, D.: Isnap: Towards intelligent tutoring in novice programming environments. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. p. 483–488. SIGCSE '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3017680.3017762, https://doi.org/10.1145/3017680.3017762
13. Rahman, M.M.: Data analysis and code assessment using machine learning techniques for programming activities (2022). https://doi.org/10.15016/00000215
14. Rahman, M.M., Watanobe, Y., Kiran, R.U., Kabir, R.: A stacked bidirectional lstm model for classifying source codes built in mpls. In: Machine Learning and Principles and Practice of Knowledge Discovery in Databases. pp. 75–89. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-93733-1_5
15. Rahman, M.M., Watanobe, Y., Kiran, R.U., Thang, T.C., Paik, I.: Impact of practical skills on academic performance: A data-driven analysis. IEEE Access **9**, 139975–139993 (2021). https://doi.org/10.1109/ACCESS.2021.3119145
16. Raychev, V., Schäfer, M., Sridharan, M., Vechev, M.: Refactoring with synthesis. SIGPLAN Not. **48**(10),

339–354 (oct 2013). https://doi.org/10.1145/2544173.2509544, https://doi.org/10.1145/2544173.2509544
17. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y.: Scratch: Programming for all. Commun. ACM **52**(11), 60–67 (nov 2009). https://doi.org/10.1145/1592761.1592779, https://doi.org/10.1145/1592761.1592779
18. Saha, R.K., Lawall, J., Khurshid, S., Perry, D.E.: On the effectiveness of information retrieval based bug localization for c programs. In: 2014 IEEE International Conference on Software Maintenance and Evolution. pp. 161–170 (2014). https://doi.org/10.1109/ICSME.2014.38
19. Schuster, M., Paliwal, K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing **45**(11), 2673–2681 (1997). https://doi.org/10.1109/78.650093
20. Wang, W., Zhi, R., Milliken, A., Lytle, N., Price, T.W.: Crescendo: Engaging students to self-paced programming practices. In: Proceedings of the 51st ACM Technical Symposium on Computer Science Education. p. 859–865. SIGCSE '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3328778.3366919, https://doi.org/10.1145/3328778.3366919
21. Watanobe, Y., Rahman, M.M., Kabir, R., Amin, M.F.I.: Identifying algorithm in program code based on structural features using cnn classification model. Applied Intelligence (2022)
22. Watanobe, Y.: Aizu online judge (2018), available: https://onlinejudge.u-aizu.ac.jp/
23. Watanobe, Y., Rahman, M.M., Matsumoto, T., Rage, U.K., Ravikumar, P.: Online judge system: Requirements, architecture, and experiences. International Journal of Software Engineering and Knowledge Engineering **32**(06), 917–946 (2022). https://doi.org/10.1142/S0218194022500346
24. Watanobe, Y., Rahman, M.M., Vazhenin, A., Suzuki, J.: Adaptive user interface for smart programming exercise. In: 2021 IEEE International Conference on Engineering, Technology & Education (TALE). pp. 01–07 (2021). https://doi.org/10.1109/TALE52509.2021.9678757
25. Wiggins, J.B., Fahid, F.M., Emerson, A., Hinckle, M., Smith, A., Boyer, K.E., Mott, B., Wiebe, E., Lester, J.: Exploring novice programmers' hint requests in an intelligent block-based coding environment. In: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. p. 52–58. SIGCSE '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3408877.3432538, https://doi.org/10.1145/3408877.3432538