DeepVS: an efficient and generic approach for source code modelling usage

Yasir Hussain, Zhiqiu Huang, Yu Zhou and Senzhang Wang

The source code suggestions provided by current integrated development environment (IDEs) are mostly dependent on static type learning. These suggestions often end up proposing irrelevant suggestions for a peculiar context. Recently, deep learning-based approaches have shown great potential in the modelling of source code for various software engineering tasks. However, these techniques lack adequate generalisation and resistance to acclimate the use of such models in a real-world software development environment. This Letter presents DeepVS, an end-to-end deep neural code completion tool that learns from existing codebases by exploiting the bidirectional gated recurrent unit (BiGRU) neural net. The proposed tool is capable of providing source code suggestions instantly in an IDE by using pre-trained BiGRU neural net. The evaluation of this work is two-fold, quantitative and qualitative. Through extensive evaluation on ten real-world open-source software systems, the proposed method shows significant performance enhancement and its practicality. Moreover, the results also suggest that DeepVS tool is capable of suggesting zero-day (unseen) code tokens by learning coding patterns from real-world software systems.

Introduction: Source code suggestion and completion are vital features of an integrated development environment (IDE). Software developers excessively rely on such features while developing software. Most of the modern IDE’s source code completion tools [1] provide the next possible source code by leveraging the information already exists in the code editor of the IDE. Due to the limited historical data available in the code editor, the IDE may not be able to provide the correct suggestions or possibly torment the developer with irrelevant suggestions.

Recently, deep learning techniques have been widely applied for various source code modelling tasks such as source code completion [2, 3], error fixing [4, 5], method naming [6] and so on. These approaches [2, 3] have shown how they can significantly improve such tools by learning from historical codebases (e.g. GitHub). However, there are various limitations and concerns:

- Most of these techniques lack to capture long-term context dependencies of source code resulting in subordinate performance.
- Most techniques are usually formed from recognised natural events and, therefore, are incapable of suggesting zero-day (unseen) code tokens.
- Most methods require a huge amount of computation power as well as time to train such models.
- Current techniques lack to illustrate, how they can assist software developers in a real-world software development environment.
- The existing works present no enlightenment, how these models can be integrated into an IDE.

To defy the above apprehensions, we propose a neural code completion tool named DeepVS. The proposed tool leverages the bi-directional neural gating mechanism to learn coding patterns from existing software systems (codebase) resulting in a significant performance boost. This Letter makes the subsequent contributions:

- An end-to-end neural code completion tool trained on ten real-world open-source software systems with over 13 million code tokens.
- The projected method is capable of providing suggestions in real-time directly in an IDE. Further, DeepVS tool is capable of providing zero-day code tokens. The DeepVS tool is publicly available at https://github.com/yasirhuxxain/DeepVS/.
- The quantitative evaluation with ten-real world open-source software systems and qualitative analysis of DeepVS tool verifies that the projected method is accurate and subordinate the existing approaches.

Approach: The projected method begins by preprocessing the codebase. First, we perform standardisation to obtain the optimal feature set. To learn real-world coding patterns, we tokenise each source code file in the codebase into multiple sequences of fixed context. Next, for the purpose of neural language model training, we vectorise these extracted contexts. Finally, we train and test the BiGRU classifier for the code completion task.

Problem statement: Given a source code file, the predictions for the next source code token could be defined as: \( Y = f(\tau) \) where \( Y \) represents the set of next source code suggestions, \( f \) represents the classification function that predicts the next possible source code tokens and \( \tau \) represents the context which is an input to the classification function. We use the source code of real-world software systems extracted from GitHub [https://www.github.com/, accessed on 1/12/2019.] to build our codebase for the purpose of neural model training and testing. The collected codebase (CB) can be formalised as: \( CB = \{SS_1, SS_2, \ldots, SS_i\} \) and \( SS_i = \{cf_1, cf_2, \ldots, cf_j\} \), where \( SS_1, SS_2, \ldots, SS_i \) represents the \( j \) software systems involved in \( CB \) and \( cf_1, cf_2, \ldots, cf_j \) represents the \( j \) code files involved in ith software systems.

Preprocessing and feature extraction: Preprocessing is a vital process in building neural language models. Following grassroots, we perform standardisation, context extraction and vectorisation.

Standardisation: To standardise CB, we remove all blank lines, inline and block-level comments from each software system. Further, we replace all literal values to their generic data types to build the optimum feature set.

Context extraction: To learn code patterns from historical software systems, each source code file in the codebase is parted into a sequence of space-separated tokens. These sequences are then subdivided into multiple sequences of fixed context (\( \tau \)).

Vectorisation: To train the neural language model, we need to build a global vocabulary system. For this intend, we replace all singleton code tokens with a special token (‘UNK’). Next, we build the vocabulary \( V = \{v_1, v_2, \ldots, v_k\} \) where \( v_1, v_2, \ldots, v_k \) represents \( k \) unique source code tokens corresponds to an entry in the vocabulary. Then, each code token is replaced with its corresponding vocabulary index (positive integer) to convert context vectors into a form that is suitable for neural language model training. The vectorised source code can be expressed as: \( SS_i = \{r_1, r_2, \ldots, r_k\} \) where \( r_1, r_2, \ldots, r_k \) represents the \( k \) number of context vectors found in ith software systems.

Bi-directional context learner: We adopt GRU [7] which is a superior type of recurrent neural network (RNN) for the purpose of code completion classification. Conventional RNN suffers from fading gradient issue, limiting it from learning long-term context dependencies. Similar to LSTM, the GRU uses gating approach that exposes full issue, limiting it from learning long-term context dependencies. Conventional RNN suffers from fading gradient issue, limiting it from learning long-term context dependencies.

The projected method begins by preprocessing the codebase. First, we perform standardisation to obtain the optimal feature set. To learn real-world coding patterns, we tokenise each source code file in the codebase into multiple sequences of fixed context. Next, for the purpose of neural language model training, we vectorise these extracted contexts. Finally, we train and test the BiGRU classifier for the code completion task.

\[ H_p(C) \approx -\frac{1}{N} \sum_{l=1}^{N} \log P(\tau | y_{l-1}^{l+1} ) \]
serve the pre-trained BiGRU locally or over a cloud and use a plugin interface for real-time classification. Fig. 1 shows the overall workflow of the proposed approach where the offline part shows the steps involved in the pre-training BiGRU classifier and the online part illustrates the tool implementation details.

Fig. 1 Overall workflow of DeepVS

Neural server: A cloud-based neural server is built to serve the pre-trained BiGRU neural language model as a centralised attendant to determine the likelihood scores for the next source code suggestions. In this work, the cloud platform [8] is selected because of its centralised nature and scalability. The neural cloud server takes source code context (τ) as input and provides most likely next source code suggestions (Y) by consulting the pre-trained BiGRU classifier.

Deep suggestion engine: As a proof of concept and demonstration purposes, we implement the plugin interface in Visual Studio Code IDE. The DeepVS tool can be triggered by using shortcut key CTRL + SPACE. By triggering DeepVS, it takes the source code prior to the current cursor position as context (τ) and requests Neural Server for classification Y = f(τ).

Evaluation: The evaluation of this work is two-fold, quantitative and qualitative. In quantitative evaluation, we assess the classification outcomes to test the performance of the proposed method and compare it with state-of-the-art methods. Metrics used to assess the quantitative efficiency of this work are top-k accuracy and mean reciprocal rank (MRR), which are mostly applied. For the qualitative evaluation of the work, we conduct an experimental study involving seven university students having software development experience. The experiment was performed with the help of a questionnaire. All the participants were demonstrated with the basic usage of the tool. The questionnaire given to the participants is shown in Table 1.

Table 1: Questionnaire used for evaluation

| Questions                                      | 1 | 2 | 3 | 4 | 5 |
|------------------------------------------------|---|---|---|---|---|
| Q1: How easy is it to set up and use the DeepVS tool? | 1 | 2 | 3 | 4 | 5 |
| Q2: Does DeepVS tool provides suggestions in real-world coding environment? | 1 | 2 | 3 | 4 | 5 |
| Q3: Does DeepVS tool performs better than IDE’s default suggestion engine? | 1 | 2 | 3 | 4 | 5 |
| Q4: I will recommend DeepVS to others. | 1 | 2 | 3 | 4 | 5 |

Dataset: For the training and testing of the proposed method, we utilise the codebase anticipated in [9, 10]. The codebase consists of ten opensource software systems (ant, cassandra, db40, jgit, poi, batik, antlr, itext, fts, maven) containing over 13M code tokens with a large vocabulary of size 145,457 (singleton removed). Table 2 shows the statistics of model training and testing sets in detail.

Table 2: Code database statistics

| Line of code (LoC) | Total code tokens | Vocab size |
|--------------------|-------------------|------------|
| training           | 1,684,158         | 12,086,347 |
| testing            | 187,129           | 1,342,927  |
| total              | 1,871,287         | 13,429,274 |

Experimental setup and results: The neural language model and cloud platform are build using Python 3.6. The training of neural language models is done by using tensorflow v1.14 on Intel(R) Xeon(R) Silver 4110 CPU 2.10 GHz with 32 cores and 128 GB of ram running Ubuntu 18.04.2 LTS operating system, equipped with NVIDIA RTX 2080 GPU. The neural language model training is offline, thus has no impact on classification time. The pre-trained BiGRU model is used for classification purpose in neural server. The Neural Server consists of a single core Intel CPU with 1 GB Ram running Ubuntu-18 ×64-bit non-GUI edition. The average classification time of the proposed approach is less than 20 ms which is fairly adequate and acceptable.

Quantitative results: In order to realistically evaluate the effectiveness of the proposed method, we compare the performance of our proposed method with other state-of-the-art methods. We choose Hindle et al. [9] and White et al. [3] work as baselines because they are related to our target task. The comprehensive results are given in Table 3. It can be witnessed that the proposed method proves noticeably better performance as compared to other baselines. We can observe that the proposed approach achieves 22.47% higher accuracy as compared to the best baseline (RNN) by ranking the correct suggestions on the first index (Top-1).

Table 3: Accuracy and MRR scores comparison

| Model         | Accuracy | MRR  |
|---------------|----------|------|
| Top-1, %      | 48.47    | 57.68|
| Top-3, %      | 59.87    | 61.78|
| Top-5, %      | 72.18    | 76.21|
| Top-10, %     | 78.00    | 89.34|

Qualitative results: As reported in Fig. 4, participants found the DeepVS tool is easy to set up and use. In Q2, all participants have agreed that DeepVS tool is capable of providing suggestions in a real-world development environment. In Q3, out of seven participants three voted strongly agree, three of them voted agree and one voted neutral. In Q4, most of the participants have agreed that they would recommend DeepVS to their peers. The illustrative examples of source code suggestion tasks are provided in Section 17.

DeepVS tool demo: To show the effectiveness of DeepVS tool for real-world usage, consider the example presented in Fig. 5 where a software developer is writing a complex program to find the transpose of a matrix. Hereby triggering (line 9) our DeepVS tool it suggests the most probable next source code tokens i, j and k where the Visual Studio Code’s default suggestion engine fails to provide any relevant suggestions instead, it torments the developer with irrelevant suggestions. Later on, at line 21 the software developer is writing a print statement. Here the most probable next source code token is column based on the given context. Hereby triggering our DeepVS tool it suggests the correct

ELECTRONICS LETTERS 11th June 2020 Vol. 56 No. 12 pp. 604–607
next source code token column at its first index whereas the IDE’s default source code suggestion tool ranks the correct suggestion on its third index. From the given examples, we can observe that the DeepVS tool is capable of suggesting zero-day code tokens and have a better understanding of context while providing suggestions.

- To the best of our knowledge, the proposed approach is the first one that enables the usage of machine/deep learning-based source code models directly in an IDE.
- We present an end-to-end deep neural code completion tool named DeepVS which verifies the practicality of the proposed approach.
- The DeepVS tool is trained on ten real-world open-source software systems with over 13M code tokens, thus capable of suggesting zero-day (unseen) code tokens by leveraging large-scale historical codebase.
- Further, DeepVS tool has better understanding of context while providing suggestions. The proposed tool ranks the suggestions effectively without tormenting the developer with unnecessary suggestion (see Fig. 6).

Conclusion: In this Letter, we presented DeepVS, an end-to-end deep neural code completion tool that leverages from the pre-trained BiGRU classifier to provide code suggestions directly in an IDE in real time. The proposed tool is trained and tested on real-world software systems with over 13M code tokens. Further, we have demonstrated the effectiveness and practicality of the proposed tool in a real-world use case (IDE). Moreover, the approach illustrated in this work is general and can help enable the usage of neural language models for other source code modelling tasks directly in an IDE.

Acknowledgments: This work was supported by the National Key R&D (grant no. 2018YFB1003902), Natural Science Foundation of Jiangsu Province (No. BK20170809), National Natural Science Foundation of China (No. 61972197, No. 61802179) and Qing Lan Project.

© The Institution of Engineering and Technology 2020
Submitted: 19 February 2020  E-first: 7 April 2020
doi: 10.1049/el.2020.0500
One or more of the Figures in this Letter are available in colour online.
Yasir Hussain, Yu Zhou and Senzhang Wang (College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, People’s Republic of China)
Zhiqiu Huang (College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, People’s Republic of China)
✉ E-mail: zqhuang@nuaa.edu.cn

Proposed work’s major benefits: The qualitative and quantitative evaluation verifies the feasibility and practicality of this work. We summarise the benefits of our proposed framework and the DeepVS tool as follows:

- Source code suggestion example with and without the DeepVs tool along with their probabilities
- Conclusion: In this Letter, we presented DeepVS, an end-to-end deep neural code completion tool that leverages from the pre-trained BiGRU classifier to provide code suggestions directly in an IDE in real time. The proposed tool is trained and tested on real-world software systems with over 13M code tokens. Further, we have demonstrated the effectiveness and practicality of the proposed tool in a real-world use case (IDE). Moreover, the approach illustrated in this work is general and can help enable the usage of neural language models for other source code modelling tasks directly in an IDE.

Acknowledgments: This work was supported by the National Key R&D (grant no. 2018YFB1003902), Natural Science Foundation of Jiangsu Province (No. BK20170809), National Natural Science Foundation of China (No. 61972197, No. 61802179) and Qing Lan Project.

© The Institution of Engineering and Technology 2020
Submitted: 19 February 2020  E-first: 7 April 2020
doi: 10.1049/el.2020.0500
One or more of the Figures in this Letter are available in colour online.
Yasir Hussain, Yu Zhou and Senzhang Wang (College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, People’s Republic of China)
Zhiqiu Huang (College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, People’s Republic of China)
✉ E-mail: zqhuang@nuaa.edu.cn

References

1. Bruch, M., Mongeur, M., and Mezini, M.: ‘Learning from examples to improve code completion systems’. Proc. of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Amsterdam, the Netherlands, 2009, pp. 213–222
2. Raychev, V., Vechev, M., and Yahav, E.: ‘Code completion with statistical language models’. ACM Sigplan Notices, Edinburgh, UK, 2014, vol. 49, pp. 419–428
3. White, M., Vendome, C., Linares-V, M., et al.: ‘Towards deep learning software repositories’. Proc. of the 12th Working Conf. on Mining Software Repositories, Florence, Italy, 2015, pp. 334–345
4. Gupta, R., Kanade, A., and Shevade, S.: ‘Deep reinforcement learning for programming language correction’, arXiv preprint arXiv:1801.10467, 2018
5. Santos, E.A., Campbell, J.C., Patel, D., et al.: ‘Syntax and sensibility: using language models to detect and correct syntax errors’. 2018 IEEE 25th Int. Conf. on Software Analysis, Evolution and Reengineering (SANER), Campanbasso, Italy, 2018, pp. 311–322
6. Alon, U., Zilberstein, M., Levy, O., et al.: ‘code2vec: learning distributed representations of code’. Association for Computing Machinery, New York, NY, USA, 2019
7. Cho, K., Van Merriënboer, B., Gulcehre, C., et al.: ‘Learning phrase representations using rnn encoder-decoder for statistical machine translation’, arXiv preprint arXiv:1406.1078, 2014
8. Armbrust, M., Fox, A., Griffith, R., et al.: ‘A view of cloud computing’. Commun. ACM, 2010, 53 (4), pp. 50–58
9. Hindle, A., Barr, E.T., Su, Z., et al.: ‘On the naturalness of software’. 2012 34th Int. Conf. on Software Engineering (ICSE), Zurich, Switzerland, 2012, pp. 837–847
10. Nguyen, A.T., Nguyen, T.D., Phan, H.D., et al.: ‘A deep neural network language model with contexts for source code’. 2018 IEEE 25th Int. Conf. on Software Analysis, Evolution and Reengineering (SANER), Campanbasso, Italy, 2018, pp. 323–334