AI for Pair Programming (MPM Proposal)

Introduction

The AI craze detonated by deep learning has driven the evolution of all walks of life, such as protein structure prediction [1], multimedia artistic creation [2, 3], and robotics intelligence simulation [4]. In the area of programming and software, exciting things have also emerged, and such one representative is GitHub Copilot [1]. It feels like your pair programmer, always making good development suggestions. However, the project itself is not publicly available, and the technical details in building the system are hidden [5]. Considering GitHub Copilot represents the direction of assisted/automated programming in the future, SEAL [2] proposes the project that developing an intelligent productivity tool connecting the programmers in the industry and researchers in academia, to promote software automation.

Approach

Overall, our approach takes the same design as the basic architecture of GitHub Copilot, as shown in 1. GitHub Copilot consists of two core components: the Codex model [3] and the GitHub Copilot service. The former is a GPT language model trained on both the code data and text data, playing the role of code synthesizer and programming suggester, while the latter is in charge of interactive activities between the Codex model and programmers, such as monitoring the user actions in the editor and optimizing the programming suggestions given by the model.

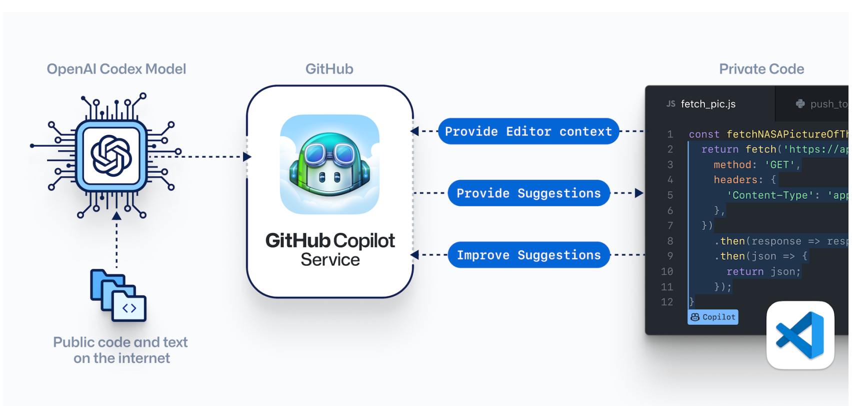

Figure 1: The schematic diagram of GitHub Copilot.

In the designed architecture, the frontend is the editor, where the language client and server run locally together as the editor extension, and the backend is the code service running in the cloud. As illustrated in 2, the language client and server are marked green while the code service and its supportive components including the neural models are marked red.

The deliverable of the project is expected to be a powerful editor extension for a specific mainstream programming language. Based on the current architecture design, the entire workload can be divided into the following missions (open to discussion or modification):

[1] https://copilot.github.com

[2] https://www.ifi.uzh.ch/en/seal/about.html

[3] https://openai.com/blog/openai-codex/

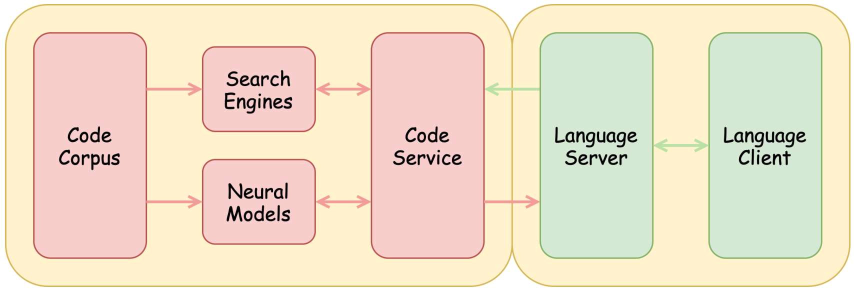

Figure 2: The architecture diagram of the intelligence system.

Developing the Language Client and Server (around 4 weeks)

In this mission, the group is to develop the language client and server. The language client monitors the edit actions of developers and sends requests to the language server whenever possible. The language server responds to the language client with intelligence supports based on the given information.

To minimize the workload for non-critical functions, it is strongly recommended to develop the client and server for Python, and with the maximum reference to the relevant projects, such as jedi, pyright, and vscode-python. The required features are syntax highlighting and type completion. The compatibility with the default language server Pylance [4] should be cared about.

Deploying the Code Service and Supportive Components (around 8 weeks)

Once the language server and client are ready, the group could consider enabling powerful features to assist programming [6], by using the code service and code corpus. The first step is to explore available code corpus such as PyTorrent [7], adopt a code formatter such as yapf or black, and store the formatted code snippets into MongoDB with extracted metadata. The second step is to deploy information retrieval engines to support code search tasks, for example, searching semantically similar code snippets for the given natural language query or code sample, where BM25 [8] and MIPS [9, 10] are recommended for sparse and intense retrieval functions. The lightweight implementation should be via Gensim and is recommended, while the heavy way is dependent on elasticsearch and faiss. The third step is to integrate existing neural models to support intelligent features, where PyART [11] and Type4Py [12] are ready for API recommendation and type inference.

Building the Model for Conditional Code Generation (around 8 weeks)

The core component inside the GitHub Copilot is the Codex Model, trained on a large amount of code data and working as the engine for code generation. The group will study the latest research work in the fields of code generation and conditional text generation, to build a neural model as the preliminary solution. Besides Codex, the relevant and representative papers are CodeT5 [13], Optimus [14], and Puzzles [15], corresponding to the models for code generation, conditional text generation, and the high-quality dataset. The novel ideas or findings in this mission could be further studied and eventually formed as a publication draft, standalone to the project.

Establishing the Feedback Loop (around 4 weeks)

When the project is completed, the deliverable should be released as an editor extension, under the name of SEAL. It would be very nice if the extension could populate among software developers, therefore the group could work a bit to advertise the editor extension.

Assume we already have certain users, the most important aspect is to establish the feedback loop [16] between programmers and researchers, so that the failed cases and drawbacks of specific features could be highlighted. Therefore, the group should find a way to introduce the feedback mechanism, sending back the usage statistics and implicit feedback from the running extension instances to the lab, as empirical evidence to inspire potential research ideas.

[4] https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance

Discussion

There are a few business solutions and closed-source tools for AI-powered assistive programming, which could be checked for a better understanding of our project. Besides GitHub Copilot, they are TabNine [5], AIXcoder [6], Kite [7], IntelliCode [8], etc.

The project should be a team project for two Master's students, with the general knowledge of software development and deep learning. The estimated duration is around six months. The project benefits the ongoing and further research projects of deep learning for software engineering (DL4SE) [17], for example, it would contribute to one case study on this topic shortly. In addition, researchers in the lab could therefore build a channel to gain experience and insights about assisted/automated programming, via the interactions with software developers or by studying the usage statistics and implicit feedback from the feedback loop.

References

[1] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[2] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *ArXiv*, abs/2102.12092, 2021.

[3] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *ArXiv*, abs/2005.00341, 2020.

[4] Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *arXiv preprint arXiv:2102.02202*, 2021.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harri Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[6] Triet Huynh Minh Le, Hao Chen, and Muhammad Ali Babar. Deep learning for source code modeling and generation. *ACM Computing Surveys (CSUR)*, 53:1 – 38, 2020.

[7] Mehdi Bahrami, Shrikanth N. C., Shade Ruangwan, Lei Liu, Yuji Mizobuchi, Masahiro Fukuyori, Wei-Peng Chen, Kazuki Munakata, and Tim Menzies. Pytorrent: A python library corpus for large-scale language models. *ArXiv*, abs/2110.01710, 2021.

[8] Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3:333–389, 2009.

[9] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *ArXiv*, abs/1405.5869, 2014.

[10] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. Quantization based fast inner product search. *ArXiv*, abs/1509.01469, 2016.

[11] Xincheng He, Lei Xu, X. Zhang, Rui Hao, Yang Feng, and Baowen Xu. Pyart: Python api recommendation in real-time. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1634–1645, 2021.

[12] Amir M. Mir, Evaldas Latoskinas, Sebastian Proksch, and Georgios Gousios. Type4py: Deep similarity learning-based type inference for python. *ArXiv*, abs/2101.04470, 2021.

[13] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *ArXiv*, abs/2109.00859, 2021.

[5]https://www.tabnine.com
[6]https://www.aixcoder.com
[7]https://www.kite.com/
[8]https://visualstudio.microsoft.com/services/intellicode

[14] Chunyuan Li, Xiang Gao, Yuan Li, Xiujun Li, Baolin Peng, Yizhe Zhang, and Jianfeng Gao. Optimus: Organizing sentences via pre-trained modeling of a latent space. In *EMNLP*, 2020.

[15] Tal Schuster, A. Kalyan, Oleksandr Polozov, and Adam Tauman Kalai. Programming puzzles. *ArXiv*, abs/2106.05784, 2021.

[16] H Giese, Y Brun, J Di Marzo Serugendo, C Gacek, H Kienle, H Müller, M Pezze, and M Shaw. Engineering self-adaptive and self-managing systems. *Software Engineering for Self-Adaptive Systems. LNCS*, 5525:47–69, 2009.

[17] Cody Watson, Nathan Cooper, David Nader-Palacio, Kevin Moran, and Denys Poshyvanyk. A systematic literature review on the use of deep learning in software engineering research. *ArXiv*, abs/2009.06520, 2020.