RepoGenix: Dual Context-Aided Repository-Level Code Completion with Language Models

Ming Liang, Xiaoheng Xie, Gehao Zhang, Xunjin Zheng, Peng Di, Wei Jiang, Hongwei Chen, Chengpeng Wang and Gang Fan
Ant Group
China

ABSTRACT

The success of language models in code assistance has spurred the proposal of repository-level code completion as a means to enhance prediction accuracy, utilizing the context from the entire codebase. However, this comprehensive context comes at a cost: while it enhances model performance, it also increases inference latency. This balance between improved accuracy and computational efficiency poses a significant challenge in real-world applications. We present RepoGenix, a solution that enhances repository-level code completion without increased latency. RepoGenix combines *analogous context* and *relevant context*, using *Context-Aware Selection* technology to efficiently compress these contexts into limited-size prompts. Our experiments on CrossCodeEval demonstrate that RepoGenix not only achieves a substantial 48.41% reduction in inference time, but also yields improvement in performance compared to baseline methods. We have successfully implemented and tested RepoGenix within AntGroup's development environments. This approach is being extended to multiple programming languages and will be open-sourced, aiming to enhance code completion efficiency for the broader developer community.

CCS CONCEPTS

• **Software and its engineering** → **Automatic programming**.

KEYWORDS

Repository-Level Code Completion, Code Language Models, Retrieval Augmented Generation

ACM Reference Format:
Ming Liang, Xiaoheng Xie, Gehao Zhang, Xunjin Zheng, Peng Di, Wei Jiang, Hongwei Chen, Chengpeng Wang and Gang Fan. 2024. RepoGenix: Dual Context-Aided Repository-Level Code Completion with Language Models. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3691620.3695331

Correspondence to: Wei Jiang <jonny.jw@antgroup.com>, Gang Fan <fangang@antgroup.com>.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695331

1 INTRODUCTION

Code language models (Code LMs) excel in programming tasks, especially code completion, potentially boosting developer productivity [1]. Trained via causal language modeling, these models predict code tokens using *in-file context*. Repository-level code completion [2, 7] leverages *cross-file context*, encompassing high-level abstractions across the codebase, to enhance code completion accuracy and relevance. This integration is particularly crucial in IDEs where both accuracy and rapid response are critical [4]

Although some works [3, 5, 7] have leveraged the Retrieval-Augmented-Generation (RAG) strategy to enhance code completion prediction accuracy, they present a challenge of increased latency. This refers to the trade-off between richer context improving predictions and longer prompt lengths increasing inference times, which can hinder performance in IDE scenarios.

To address the trade-off between context and latency, we propose a dual context-aided RAG methodology that efficiently captures comprehensive contextual information within a limited token space. Inspired by human cognition, our approach synthesizes two key types of cross-file context: the *Analogous Context(AC)*, derived from code similarity analysis to identify functionally similar code segments, and the *Relevant Context(RC)*, which provides semantic understanding of class taxonomies and API interactions. These complementary contexts collaborate to offer a comprehensive codebase understanding. RepoGenix also utilizes *Context-Aware Selection (CAS)* to filter the most relevant context, condensing it into a limited-size prompt, thereby reducing inference latency.

We evaluate RepoGenix using CrossCodeEval[2] benchmark. RepoGenix achieves a significant 48.41% reduction in inference time while also providing improvement in performance compared to baseline methods. This enhanced efficiency makes RepoGenix particularly well-suited for fast-response development environments, such as IDE integration.

2 REPOGENIX IN A NUTSHELL

To illustrate the core functionality of RepoGenix, let's delve into fig. 1 as an example. RepoGenix functions seamlessly through the following three critical stages:

Relevant Context Extraction. To construct the relevant context Γ_{rc} effectively, we extend the traditional Code Property Graph [6] by focusing specifically on extracting relevant context from the source code for code completion tasks within repositories. For instance, as illustrated in fig. 1, we identify three code entities with import relationships pertinent to the code snippet shown in fig. 1(a) and fig. 1(b). By understanding the intricate relationships and dependencies among various code entities, the model can make more informed and contextually appropriate predictions.

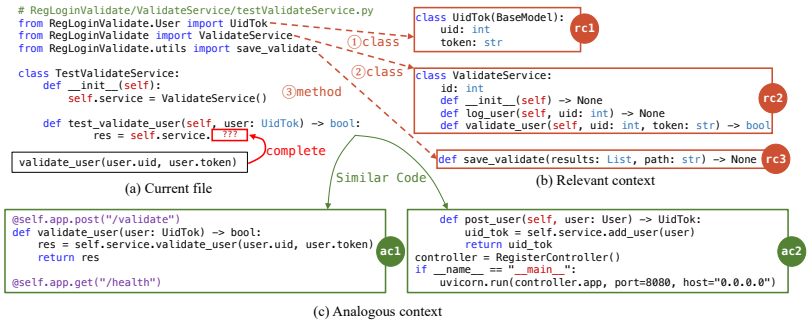

Figure 1: Illustrative example of relevant context and analogous context in use.

Methods	Prompt Length	ES	ID-F1	SpeedUp
Analogous Context	4096	78.65	74.19	0%
Relevant Context	4096	78.69	74.85	0%
RepoGenix w.o CAS	4096	80.82	77.31	0%
RepoGenix with CAS	2048	80.27	76.36	33.29%
RepoGenix with CAS	1024	79.73	75.29	48.41%

Table 1: Results on Python subset using StarCoder-7B.

Analogous Context Extraction. An analogous context Γ_{ac} is a set of code chunks, where a code chunk $ck \in \Gamma_{ac}$ is highly similar to an unfinished code chunk ck^* in the currently edited file. To compute the analogous context, we analyze the currently edited file to identify unfinished code chunks (ck^*). We then discover several similar code chunks in other source files, whose successors provide informative guidance for completion. In fig. 1(c), the analogous context is derived based on the content of the current file (fig. 1(a)). Specifically, the code chunks $ac1$ and $ac2$ share similar tokens with the current file, such as "user" and "service." These high similarities indicate that the chunks in the analogous context should be prioritized during the prompting process, as they are likely to exhibit similar or identical functionalities to the code in the current file.

Context-Aware Selection. To combine relevant context and analogous context into a fixed-length prompt, we designed the Context-Aware Selection(CAS) technique. Theoretically, reliance on post-execution metrics such as the Edit Similarity (ES) value for evaluating code completions often fails to provide actionable insights in real-time scenarios, as these metrics only assess outcomes after all computations are completed. Therefore, CAS introduces the relevance score r^*_{ck} as a proactive indicator that can approximate the value of the outputs before the actual code completion occurs. In the example shown in fig. 1, our CAS strategy selects rc_1, rc_2, and ac_1 to create the optimized dual context, as other contexts do not contribute to the current completion task. Through CAS, we filter out irrelevant information while reducing the prompt length, thereby enhancing inference efficiency.

3 EVALUATION

We evaluate RepoGenix using the CrossCodeEval benchmark python dataset [2], which is designed to evaluate code completion frameworks at the repository level, the content to be completed includes at least one in-file API reference within the repository. Table 1 illustrates our main results. The data clearly demonstrates that RepoGenix (without CAS), which utilizes both relevant and analogous context simultaneously, significantly outperforms using either context type alone when prompt length is unrestricted. Specifically, compared to using only Relevant Context, RepoGenix shows improvements of 2.13 and 2.46 in the ES metric and ID-F1 score. When compared to using only Analogous Context, the improvements are 2.17 and 3.12, respectively. These results strongly suggest that the two sources of context information are indeed complementary.

Moreover, when applying the CAS technique to limit prompt length, RepoGenix still outperforms both the only relevant and only analogous approaches in terms of ES and F1-Score, while achieving substantial speed improvements of 33.29% and 48.41% with prompt length of 2048 and 1024 respectively. The selection of prompt size for RepoGenix presents a trade-off: smaller prompt sizes lead to significant speedup at the cost of minor performance degradation, which will be determined in real world.

REFERENCES

[1] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA1 (2023), 85–111. https://doi.org/10.1145/3586030

[2] Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. 2024. CROSSCODEEVAL: a diverse and multilingual benchmark for cross-file code completion. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2023, 23 pages.

[3] Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. 2023. Repository-Level Prompt Generation for Large Language Models of Code. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 1314, 23 pages.

[4] Chaozheng Wang, Junhao Hu, Cuiyun Gao, Yu Jin, Tao Xie, Hailiang Huang, Zhenyu Lei, and Yuetang Deng. 2023. How Practitioners Expect Code Completion?. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (, San Francisco, CA, USA,) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1294–1306. https://doi.org/10.1145/3611643.3616280

[5] Chengpeng Wang, Yixiao Yang, Han Liu, and Le Kang. 2019. Statistical api completion based on code relevance mining. In *2019 IEEE Workshop on Mining and Analyzing Interaction Histories (MAINT)*. IEEE, 7–13.

[6] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. 2014. Modeling and Discovering Vulnerabilities with Code Property Graphs. In *2014 IEEE Symposium on Security and Privacy*. 590–604. https://doi.org/10.1109/SP.2014.44

[7] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 2471–2484. https://doi.org/10.18653/v1/2023.emnlp-main.151