# RepoSim: Evaluating Prompt Strategies for Code Completion via User Behavior Simulation

Chao Peng
pengchao.x@bytedance.com
ByteDance
Beijing, China

Qinyun Wu
wuqinyun@bytedance.com
ByteDance
Beijing, China

Jiangchao Liu
liujiangchao@bytedance.com
ByteDance
Hangzhou, China

Jierui Liu
liujierui.0723@bytedance.com
ByteDance
Hangzhou, China

Bo Jiang
jiangbo.jacob@bytedance.com
ByteDance
Shenzhen, China

Mengqian Xu*
xmq@stu.ecnu.edu.cn
East China Normal University
Shanghai, China

Yinghao Wang
wangyinghao.ahab@bytedance.com
ByteDance
Shanghai, China

Xia Liu
linlandong@bytedance.com
ByteDance
Shenzhen, China

Ping Yang
yangping.cser@bytedance.com
ByteDance
Beijing, China

## ABSTRACT

Large language models (LLMs) have revolutionized code completion tasks. IDE plugins such as Copilot can generate code recommendations, saving developers significant time and effort. However, current evaluation methods for code completion are limited by their reliance on static code benchmarks, which do not consider human interactions and evolving repositories. This paper proposes RepoSim, a novel benchmark designed to evaluate code completion tasks by simulating the evolving process of repositories and incorporating user behaviors. RepoSim leverages data from an IDE plugin, by recording and replaying user behaviors to provide a realistic programming context for evaluation. This allows for the assessment of more complex prompt strategies, such as utilizing recently visited files and incorporating user editing history. Additionally, RepoSim proposes a new metric based on users' acceptance or rejection of predictions, offering a user-centric evaluation criterion. Our preliminary evaluation demonstrates that incorporating users' recent edit history into prompts significantly improves the quality of LLM-generated code, highlighting the importance of temporal context in code completion. RepoSim represents a significant advancement in benchmarking tools, offering a realistic and user-focused framework for evaluating code completion performance.

## KEYWORDS

Code Completion, Prompt Engineering, Benchmark, Large Language Model

*Work done during internship at ByteDance.


*Conference acronym 'XX, June 03–05, 2018, Woodstock, NY*

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

**ACM Reference Format:**
Chao Peng, Qinyun Wu, Jiangchao Liu, Jierui Liu, Bo Jiang, Mengqian Xu, Yinghao Wang, Xia Liu, and Ping Yang. 2018. RepoSim: Evaluating Prompt Strategies for Code Completion via User Behavior Simulation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

Large language models (LLMs) have spurred numerous innovative applications, particularly in automatic code completion tasks. Integrated Development Environment (IDE) plugins like Copilot [7] and CodeWhisperer [8] can generate code recommendations in real-time as developers write code, potentially saving up to 26 26% of developers' time and effort on coding [5]. Recognizing the immense value of LLM-driven auto-completion, researchers have extensively explored this domain, leading to the rapid development of new models and prompting strategies. Consequently, a systematic evaluation method for these models and strategies is essential to facilitate their improvement and ensure their practical applicability in real-world coding environments.

Popular evaluation datasets for LLMs, such as HumanEval [3] and DS-1000 [13], focus on code generation of standalone functions at the single-file level, making the first step towards assessing LLMs' capabilities in implementing algorithms and solving problems. To evaluate code completion in multi-file repositories, researchers have proposed repository-level benchmarks for code completion tasks, such as RepoFusion [20] and RepoBench [17] which provide context retrieved from the repositories.

However, current evaluation methods use *static* code as a benchmark and do not consider *human behaviors*. This gap makes it difficult to comprehensively evaluate code completion tasks due to the following challenges.

**Challenge 1: Construct of realistic prompts.** In practice, prompts for code completion models can leverage various pieces of information available in the IDE, such as code from all opened or recently opened files, edit history, and contents in the current file,

to improve the quality of generated code [11, 19]. Evaluating this strategy is infeasible with a static repository because it lacks the necessary context of open files during code editing.

Furthermore, existing benchmarks construct prompts using the prefix and suffix of the ground truth code, as well as contents from other files in the repository. However, this information can be *different* from what is present during real-time code completion. For instance, the suffix may be further edited before it is submitted to the repository, making the prompts less realistic.

**Challenge 2: Evaluation Metrics.** Another challenge is assessing the output of code completion tools, as the decision to accept or reject a prediction is inherently a human behavior. Existing benchmarks use the similarity to the ground truth code or the pass rate against pre-defined test cases to evaluate the correctness of the generated code [1, 3, 4, 6, 9, 10, 13–17, 21, 22]. However, to the best of our knowledge, there is no empirical study and clear evidence that these metrics accurately reflect human preferences and are positively related to the acceptance rate of human developers.

In this paper, we introduce RepoSim, a novel approach for evaluating code completion tasks by simulating the evolving process of repositories and considering user behaviors. We utilize a code completion IDE plugin deployed within our collaborated company, actively used by thousands of developers daily. The simulation is conducted by recording and replaying user behaviors during programming sessions. RepoSim stands out from existing benchmarks in two key aspects:

- **Prompt Strategy.** It provides the actual programming context in which the code to be generated is written, enabling the evaluation of more prompt strategies (such as recently visited files);
- **Evaluation Metrics.** By observing users' acceptance or rejection behaviors towards the plugin's predictions, we propose a new metric that better aligns with users' preferences compared to current evaluation criteria;

These unique features make RepoSim a valuable tool for assessing code completion performance in a more realistic and user-centric manner.

Our preliminary experimental result reveals several prompt strategies that significantly improve the quality of predicted code by instructing LLMs to make more reasonable predictions. For example, providing the model with prefix and edit history is more effective than with the prefix and suffix, while the later is widely used by current benchmarks.

Overall, the main contributions of the paper are:

- We introduce RepoSim, a simulation framework that can simulate the evolving process of a repository by recording and replaying user behaviors. This approach provides a more realistic evaluation of code completion tasks;
- By observing real users' acceptance or rejection of predicted code, we propose a new metric that better aligns with users' preferences compared to current evaluation criteria.
- Using RepoSim, we evaluated both prompt strategies and models, concluding that incorporating the user's most recent editing history can significantly improve the quality of the completed code by LLMs. This suggests that the code to be completed is not only related to its neighbors in space, but also to its neighbors in time;

# 2 BACKGROUND AND RELATED WORK

## 2.1 Large Language Models for Code

Code completion is a critical component in enhancing productivity of modern software Development. These task involves automatically generating code snippets or completing existing code based on given inputs or contexts. *General-purpose LLMs*, such as GPT-4, have demonstrated exceptional performance across various development tasks, including code generation. These models, trained on diverse datasets, can generate code snippets, complete functions, and even provide debugging assistance based on natural language prompts. Their high pass rates on benchmarks such as HumanEval highlight their capabilities in producing accurate and functional code. *Specialized LLMs*, designed primarily for code-related tasks, often outperform general models in generating precise and contextually appropriate code. Examples include Codex, DeepSeek Coder, and StarCoder, which are fine-tuned to handle code-specific data using techniques like next-token prediction and filling-in-the-middle (FIM). These models excel in producing accurate code completions and generating new code snippets within given contexts.

## 2.2 Evaluation of Code Completion

Evaluating LLMs is crucial for understanding their capabilities, especially given their black-box nature. Benchmarks like HumanEval [3] and MBPP [1] assess models on relatively simple Python functions, while more advanced benchmarks such as APPS [10] and ClassEval [6] extend this to more complex problems and class-level code generation. However, these benchmarks typically assess models on isolated tasks without considering the broader context of real-world coding environments.

Recent benchmarks, such as CrossCoderEval [4], RepoBench [17], and RepoEval [22], focus on repository-level tasks, including code completion and project-oriented evaluations. Despite their advancements, these benchmarks do not consider human behavior necessary for thorough evaluation. Thus, the need for more robust evaluation methods persists.

These benchmarks offer standardized tasks and metrics to compare the capabilities of different models, typically consisting of a natural language description as the input (prompt) and the corresponding code as the ground truth output. Metrics such as exact match, code similarity methods, and passing rate (Pass@k), which executes the model output against test cases to assess correctness, are commonly used.

# 3 APPROACH

In this section, we discuss our proposed framework for simulating and evaluating code completion systems. Our approach aims to push the boundaries of current practices by proposing a novel method to closely replicate real-world programming environments. We outline the architecture of our simulation system in Section 3.1, explain how we simulate the online environment in Section 3.3, and present our unique evaluation metrics in Section 3.4.

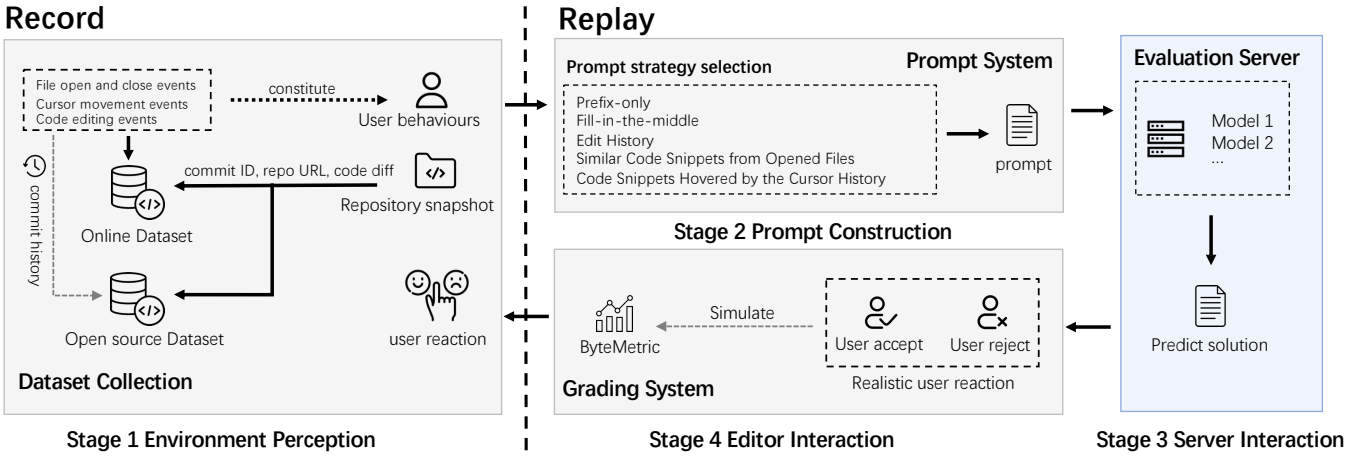


**Figure 1: Architecture**

## 3.1 System Architecture

Our simulation system, RepoSim, is designed to mimic the real-world operation of a code completion plugin. As depicted in Figure 1, the system consists of four sequential stages: environmental perception, prompt construction, server interaction, and editor interaction.

(1) **Environmental Perception Stage** gathers valuable information from the active IDE, such as code snippets from working directories and IDE events.
(2) **Prompt Construction Stage** generates prompts for code completion tasks using information obtained from the environmental perception stage.
(3) **Server Interaction Stage** facilitates communication between the IDE and the server hosting LLMs, sending requests (prompts) and receiving results (generated code).
(4) **Editor Interaction Stage** displays the results in the editor, allowing users to interact with them by accepting or rejecting the suggestions.

## 3.2 Prompt Strategies

Our approach stands out because it combines realistic simulation using actual code from the code completion plugin with the direct integration of evaluated prompt strategies into the plugin, significantly reducing effort and improving practical applicability. Supported prompt strategies include:

- **Prefix-only** uses only the preceding code (prefix) of the input cursor as the prompt. This straightforward approach serves as our baseline prompt strategy.
- **Fill-in-the-middle** uses both the prefix and suffix in prompts, offering a more realistic approach by providing better context for code completion.
- **Edit History** incorporates the user's edit history before the prefix. Edit history is represented as a git diff and ranked based on file activity (recent edited ranked first).
- **Similar Code Snippets from Opened Files** strategy adds to the prompt similar code from opened files, retrieved using the fastText [2, 12] algorithm for similarity measurement.
- **Code Snippets Hovered by the Cursor History** strategy includes lines of code where the cursor hovered before the code completion trigger point in the prompt.
- **Mixed Strategies** allow the user to select a list of strategies from above to study the effectiveness of combining different prompt strategies.

It is important to note that the prefix, suffix and code snippets collected by RepoSim differs from those in existing benchmarks, as RepoSim records in real-time when the programmer writes the code and triggers code completion. In contrast, these contents may have been altered before they are submitted to the repository where other benchmarks collect data, making them less realistic.

In addition, as the performance may vary on the number of the lines gathered from these strategies, the length of the prompt can also be configured while using RepoSim.

## 3.3 Record and Replay

To capture users' behaviors and collect data to construct prompts, we use a code completion plugin prototype that reports IDE events, including file open/close, cursor movements, code editing, and user reactions to code completion suggestions.

Instead of sending entire snapshots of repositories at the time of editing, our plugin sends the code diff with the previous commit ID to the server. This allows us to reconstruct repository snapshots at each editing point by replaying the recorded users' editing actions.

During the replaying phase, we simulate the evolution of code changes and corresponding user behaviors. By fetching the repository with the given commit ID and applying the code diff, we fully recover the project snapshot at each point. Replaying users' behaviors determines opened files and cursor positions, restoring the complete IDE environment.

Our simulation system enables two key tasks: comparing LLMs' abilities in real deployment scenarios and evaluating different prompt strategies. Each code change is treated as a code completion task, allowing LLMs to be tested by replacing the server address. Various prompt strategies can be compared by adding potential context

to the prompt before sending a request to the server-based model, determining the best strategy for user acceptance.

### 3.4 Evaluation Metrics

We propose a novel evaluation metrics, ByteMetric, based on classification model, addressing the limitations of existing metrics such as code similarity and unit testing which ignore semantics of code and do not necessarily related to the user acceptance and rejection behavior.

ByteMetric leverages online data and utilizes features such as code structure match, longest common sequence match, and users' acceptance or rejection behavior towards predictions and corresponding real editing codes. These features are used to train a classification model, with users' acceptance or rejection behavior serving as the label for the model. The resulting model serves as a metric to predict how humans would react when evaluating LLMs in the RepoSim testing system.

## 4 EMPIRICAL EVALUATION

In this section, we present experiment design and preliminary results to assess the effectiveness of RepoSim in exploring prompt strategies and models for code completion.

### 4.1 Research Questions

Our study is guided by the following research questions:

**Q1. Effectiveness of the Simulation System:** *Does this simulation system help find good prompts and achieve better acceptance rate than other prompt strategies?*
To answer this question, we use prompts produced by different prompt strategies as summarized in Section 3.2 to generate code snippets for open-source and industrial repositories. We compare the acceptance rate of these strategies.

**Q2. Model Performance:** *How do different code completion models perform with RepoSim?*
To answer this question, we use open-source and commercial models to generate code snippets with different prompt strategies and assess the achieved acceptance rate. We expect to gain insights on model training improvements such as training data cleaning, filtering and augmentation to include more diverse code and repository data.

**Q3. Evaluation Metrics:** *To what extent, the evaluation metrics are aligned with human accept and reject behaviors?*
The evaluation metrics are used to predict human acceptance or rejection of code snippets generated by LLMs. Therefore, we investigate the accuracy and recall rate of the metrics to make sure they are able to serve as the ground truth for the automated evaluation of different prompt strategies.

### 4.2 Preliminary Results

We conducted the initial experiment by collecting prefix, suffix and edit history using the prototype of the code completion plugin from developers' daily usage in our collaborated company. Our collected dataset includes 1053 code completion tasks in Python, 884 in Go, 725 in JavaScript and 975 in TypeScript.

Table 1 provides a comparative evaluation of different prompt strategies on StarCoder2 [18]. We use the strictest metrics, exact match, for this experiment.

**Table 1: Experimental Results**

| Language | Prompt Strategy | Exact Match |
|---|---|---|
| Python | Prefix | 0.17 |
| | **Prefix + Edit History** | **0.27** |
| | Prefix + Suffix | 0.17 |
| | Prefix + Suffix + Edit History | 0.24 |
| Go | Prefix | 0.21 |
| | **Prefix + Edit History** | **0.29** |
| | Prefix + Suffix | 0.20 |
| | Prefix + Suffix + Edit History | **0.29** |
| JavaScript | Prefix | 0.22 |
| | **Prefix + Edit History** | **0.3** |
| | Prefix + Suffix | 0.19 |
| | Prefix + Suffix + Edit History | 0.27 |
| TypeScript | Prefix | 0.21 |
| | **Prefix + Edit History** | **0.33** |
| | Prefix + Suffix | 0.2 |
| | Prefix + Suffix + Edit History | 0.32 |

The experimental results consistently demonstrate that the *Prefix + Edit* History strategy significantly outperforms the other prompt strategies across all evaluated programming languages and showed the highest Exact Match scores. Comparing with the *Prefix-only* and *Prefix + Suffix* strategies, the *Prefix + Suffix + Edit History* strategy also performed better, suggesting that combining suffix information with edit history can further improve performance but not as significantly as edit history alone.

To further validate the effectiveness of edit history, our future research should include a comparison with *Prefix + Edit History* and strategies involing *Similar Code Snippets from Opened Files*. In addition, the ByteMetric and real user acceptance rate should also be measured on the dataset and compared with exact match to study the effectiveness of our proposed classification-based metrics.

## 5 CONCLUSION AND FUTURE WORK

In this paper, we introduced RepoSim, a novel simulation framework designed to evaluate code completion tasks by considering the evolving process of repositories and user behaviors. RepoSim addresses the limitations of existing benchmarks by providing a realistic programming context through the recording and replaying of user behaviors, enabling the assessment of more complex prompt strategies and user-centric evaluation metrics. Our preliminary results demonstrate that incorporating users' recent edit history into prompts significantly improves the quality of LLM-generated code, highlighting the importance of temporal context in code completion.

The promising results from our preliminary evaluation of RepoSim open several directions for future research:

- **Extending the Dataset and User Base:** Expanding the dataset by including more diverse user interactions such as opened files and contents hovered over by the cursor.

- **Evaluation on ByteMetric:** Future work will involve training the ByteMetric model and conducting more extensive experiments to validate the proposed classification-based metrics in the ability of predicting user acceptance and rejection behaviors.
- **Exploring Hybrid Prompt Strategies with More Models:** Investigating the combination of multiple prompt strategies, such as integrating edit history with context from recently opened files or cursor-hovered snippets, can further enhance the performance of code completion models. Exploring these hybrid strategies on different models will also provide deeper insights into their practical applicability and model traning insights.
- **Collaboration with Industry Partners:** Partnering with software development companies to deploy RepoSim in their development workflows will provide valuable industry-specific insights. These collaborations can help tailor RepoSim to meet the unique needs of different development teams and projects, further validating its utility and impact in diverse, real-world settings.

By pursuing these directions, we aim to refine RepoSim into a robust, versatile tool that not only benchmarks prompt strategies and code completion models effectively but also drives the development of more intelligent and context-aware code completion systems. We believe that RepoSim has the potential to transform the evaluation landscape for code completion, ultimately leading to more efficient and productive coding experiences for developers.

## REFERENCES


[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

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

[4] Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, et al. 2024. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *Advances in Neural Information Processing Systems* 36 (2024).

[5] Thomas Dohmke. 2023. The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot. http://web.archive.org/web/20230627162100/https://github.blog/2023-06-27-the-economic-impact-of-the-ai-powered-developer-lifecycle-and-lessons-from-github-copilot/. Accessed: 2023-08-14.

[6] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2023. Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation. *arXiv preprint arXiv:2308.01861* (2023).

[7] Nat Friedman. 2021. Introducing GitHub Copilot: your AI pair programmer. https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer. Accessed: 2023-08-14.

[8] Nat Friedman. 2023. AI Code Generator - Amazon CodeWhisperer. https://aws.amazon.com/codewhisperer/. Accessed: 2023-08-14.

[9] Yiyang Hao, Ge Li, Yongqiang Liu, Xiaowei Miao, He Zong, Siyuan Jiang, Yang Liu, and He Wei. 2022. Aixbench: A code generation benchmark dataset. *arXiv preprint arXiv:2206.13179* (2022).

[10] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938* (2021).

[11] Maliheh Izadi, Jonathan Katzy, Tim Van Dam, Marc Otten, Razvan Mihai Popescu, and Arie Van Deursen. 2024. Language Models for Code Completion: A Practical Evaluation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[12] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, 427–431.

[13] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*. PMLR, 18319–18345.

[14] Jia Li, Ge Li, Xuanming Zhang, Yihong Dong, and Zhi Jin. 2024. EvoCodeBench: An Evolving Code Generation Benchmark Aligned with Real-World Code Repositories. *arXiv preprint arXiv:2404.00599* (2024).

[15] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science* 378, 6624 (2022), 1092–1097.

[16] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems* 36 (2024).

[17] Tianyang Liu, Canwen Xu, and Julian McAuley. 2023. RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems. arXiv:2306.03091 [cs.CL]

[18] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. StarCoder 2 and The Stack v2: The Next Generation. *arXiv preprint arXiv:2402.19173* (2024).

[19] Anton Semenkin, Yaroslav Sokolov, and Evgeniia Vu. 2024. Context Composing for Full Line Code Completion. *arXiv preprint arXiv:2402.09230* (2024).

[20] Disha Shrivastava, Denis Kocetkov, Harm de Vries, Dzmitry Bahdanau, and Torsten Scholak. 2023. RepoFusion: Training Code Models to Understand Your Repository. arXiv:2306.10998 [cs.LG]

[21] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.

[22] Fengji Zhang, Bei Chen, Yue Zhang, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. *arXiv preprint arXiv:2303.12570* (2023).