ContextPilot: Code Context Engineering with Memory-Augmented Exploration Agents

Shuzheng Gao
The Chinese University of Hong Kong
Hong Kong, China
szgao23@cse.cuhk.edu.hk

Chaozheng Wang
The Chinese University of Hong Kong
Hong Kong, China
czwang23@cse.cuhk.edu.hk

Shuqing Li
The Chinese University of Hong Kong
Hong Kong, China
sqli21@cse.cuhk.edu.hk

Yun Peng
The Chinese University of Hong Kong
Hong Kong, China
ypeng@cse.cuhk.edu.hk

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong, China
lyu@cse.cuhk.edu.hk

ABSTRACT

Large language models (LLMs) have been widely adopted to address real-world repository-level software engineering tasks. However, such tasks pose significant challenges for LLMs due to the need to reason over vast contexts spanning multiple files and modules. Code context engineering, the process of identifying relevant code context, plays a pivotal role, yet existing approaches face substantial limitations. Retrieval-based methods rely on static strategies that often fail to capture enough information. Agent-based methods, while effective, typically depend on large models for end-to-end task solving, incurring considerable computational costs. Moreover, the extensive interactions in agent-based methods lead to lengthy input sequences, which not only increase computational overhead but also impair model's ability to leverage prior information. To address the limitations, we propose ContextPilot, a novel framework that effectively and efficiently discovers code context for complex repository-level software engineering tasks. ContextPilot comprises three key components. First, it employs a decoupled explorer-generator agent structure where a lightweight explorer model efficiently navigating repositories and a large proficiency model then generates the final answer using the collected context. Second, a tool-based memory mechanism periodically converts exploration history into concise textual summaries, ensuring the model's input context remains within a manageable length. Third, we introduce a reinforcement learning-based training method augmented with cold start distillation to optimize the explorer's capabilities in context exploration and memory management. We conduct preliminary experiments on repository-level code generation and question answering benchmarks, DevEval and LongCodeBench. The results demonstrate that ContextPilot achieves competitive or even superior performance compared to large model-based agent methods while substantially reducing computational costs.

LLM4Code '26, April 12, 2026, Rio de Janeiro, Brazil

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Shuzheng Gao, Chaozheng Wang, Shuqing Li, Yun Peng, and Michael R. Lyu. 2026. ContextPilot: Code Context Engineering with Memory-Augmented Exploration Agents. In *The Third International Workshop on Large Language Models for Code (LLM4Code '26), April 12, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Repository-level code tasks, such as code generation [7], question answering over large codebases [10], and automated issue resolution [14], are critical to modern software development. Recent research on large language models (LLMs) for software engineering has progressively evolved from simple function-level tasks to more realistic repository-level scenarios that reflect real-world engineering workflows. However, effectively handling repository-level tasks requires models to reason over vast contexts spanning multiple files, modules, and dependencies, which is a critical challenge that relies on effective code context engineering [4, 11, 12].

Code context engineering, the process of identifying relevant code context from large repositories, has emerged as a critical challenge. It requires systematically and dynamically identifying relevant contextual information tailored to different tasks. Early approaches primarily employ retrieval-based methods to locate relevant code context. For example, RepoCoder [15] uses iterative similarity-based retrieval to augment prompts. while Reporformer [13] trains models for selective context identification. Despite these advancements, retrieval-based methods demonstrate limited performance, as they rely on static strategies that often fail to capture sufficient necessary context [4]. Moreover, these methods typically require task-specific design and struggle to generalize across different scenarios.

Agent-based approaches have demonstrated superior capabilities through iterative repository interactions. For example, SWE-agent [14] employs agent-computer interfaces for automated issue resolution, while CodeAgent [16] integrates specialized tools for multi-step reasoning. These methods enable LLMs to navigate across different files using various tools, yielding more comprehensive code context. However, such agent-based methods rely heavily on large models for end-to-end task solving due to their complex tool invocations and solution generation requirements, which incurs substantial computational costs. Furthermore, extensive interactions result in accumulated lengthy input sequences,

which not only increases computational overhead but also impairs the model's ability to effectively leverage prior information.

To address these limitations, we propose ContextPilot, a novel framework for code context engineering that effectively and efficiently discovers relevant repository context. Our framework comprises three key components. First, we design a decoupled architecture consisting of two models: a small explorer model that serves as a context discovery agent to navigate codebases and identify task-relevant code, and a large proficiency model that generates the final solution using the collected context. Second, we introduce a tool-based memory mechanism that periodically converts exploration history into concise textual summaries, ensuring the input context does not continuously grow and remains within a manageable length. Third, we develop a training methodology that combines cold-start training using trajectories distilled from large models with reinforcement learning to further optimize the explorer model's capabilities in context discovery and memory management. To evaluate the performance of ContextPilot, we conduct experiments on repository-level code generation and question answering tasks. We evaluate on DevEval [7] and LongCodeBench [10] benchmarks. Compared with RAG-based method, ContextPilot achieves substantial improvements of 28.1% and 3.4% with respect to pass@1. Besides, compared with agentic methods, ContextPilot achieves comparable and even better results with large models-based end-to-end agent method and reduce the overall cost by 94.8% and 79.7% on DevEval and LongCodeBench-QA, respectively.

In summary, our contributions are as follows:

- **This work is a pioneering study exploring automated context engineering for repository-level code tasks.** We investigate approaches to automatically discover and leverage contextual information in such software engineering tasks.
- **We propose ContextPilot, a novel framework with three key components for effective and efficient code context engineering.** Our approach consists of: (1) a decoupled explorer-generator architecture, (2) a tool-based memory mechanism, and (3) a reinforcement learning-based model training method.
- **Preliminary experiments demonstrate improvements in both effectiveness and efficiency across multiple benchmarks.** Our method achieves competitive and even better performance on code generation and question answering tasks, while reducing costs by 94.8% and 79.7% compared to agentic method using large models.

2 RELATED WORK

2.1 LLMs for Software Engineering

Recent advancements in LLMs have transformed software engineering, with advanced models demonstrating remarkable capabilities across various programming tasks. ChatGPT [3] and GPT-4 [9] have shown strong performance in code generation and debugging through conversational interfaces. Claude series [2] models excel in understanding complex programs. With the development of LLMs, recent research has shifted to repository-level tasks mirroring real-world SE workflows. For example, SWE-bench [6] evaluates models on 2,294 real GitHub issues and pull requests, testing their ability to resolve authentic software engineering problems. DevEval [7] assesses cross-file code generation across Python and Java repositories. These benchmarks reveal that while LLMs excel at isolated tasks, repository-level scenarios pose challenges for context handling to manage inter-file dependencies and large codebases.

2.2 Context Engineering in SE tasks

Context engineering, the process of identifying and organizing relevant code context from large repositories, is crucial for repository-level SE tasks. Early retrieval-based methods focus on augmenting LLM prompts with relevant snippets. RepoCoder [15] uses iterative similarity-based retrieval for code completion. GraphCoder [8] leverages graph-based knowledge to combine repository-specific and general information. However, these static methods often fail to capture enough context information due to the complexity of code repository. Agent-based approaches provide dynamic alternatives through iterative repository interaction. SWE-agent [14] enables autonomous exploration in GitHub repositories via agent-computer interfaces. Though effective, agent-based methods face the performance-efficiency trade-offs: smaller agents lack reasoning depth, while larger ones incur high costs and context explosion during extended interactions. Besides, uncontrolled context expansion can also impair model's ability to comprehend and leverage previous information in the conversation.

3 METHODOLOGY

Figure 1 illustrates the overall architecture of ContextPilot, which comprises three key components working in synergy to enable efficient and effective automated context engineering. The framework employs a decoupled structure that separates context exploration from answer generation, uses a tool-based memory mechanism to manage context dynamically, and leverages a reinforcement learning-based training method to optimize the explorer's capabilities in context exploration and memory management. In the following subsections, we introduce each component in detail.

3.1 Decoupled Explorer-Generator Architecture

As shown in Figure 1, ContextPilot adopts a decoupled architecture that aims to effectively and efficiently find relevant context for generating accurate answers to repository-level tasks. The architecture separates the context exploration from the answer generation, and consists of two specialized models: a small explorer model LLM_e that serves as an automated context discovery agent, and a proficiency model M_g that generates the final answer using the collected context.

The explorer model is designed to efficiently navigate code repositories through multi-turn interactions. Formally, the exploration process can be represented as a sequence:

$$C_t = \langle p, \{r_1, a_1, o_1\}, \{r_2, a_2, o_2\}, \ldots, \{r_{t-1}, a_{t-1}, o_{t-1}\}\rangle,$$

where p denotes the initial prompt, and for each turn i, r_i represents the reasoning step, a_i the action taken, and o_i the observation received. In each turn, the model first generates a natural language reasoning step before invoking the tools. We develop a scaffold with three simple and general-purpose context exploration tools to enable model to explore diverse repositories without domain-specific customization. Specifically, we employ the following tools:

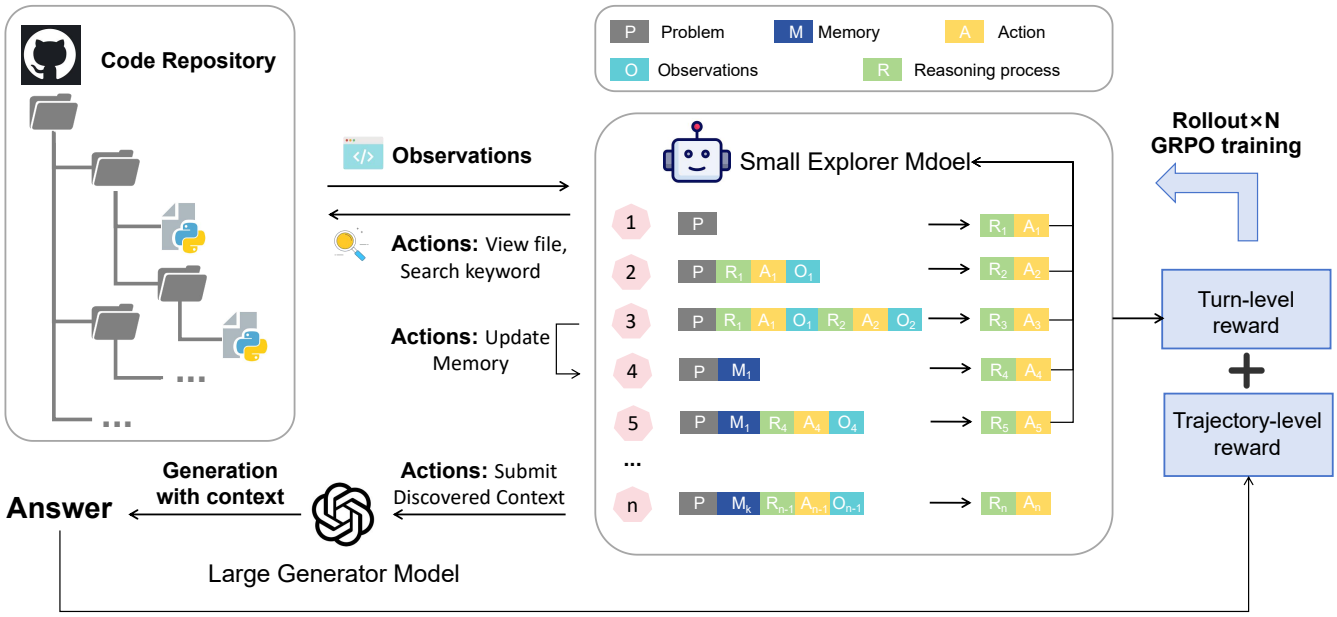

Figure 1: Overview of ContextPilot.

- **View Tool** which allows the model to inspect specific portions of a file. It takes two parameters: the filename and the viewing range (start line number l_s and end line number l_e). The tool returns the content within the specified range.
- **Search Tool** which enables keyword-based searching within a given directory. It accepts a search keyword k and a starting address a (e.g., a directory path). The tool traverses the directory tree from a and returns a list of matching files, including their full paths, line numbers where k appears, and the filenames.
- **Submit Tool** which is invoked when the model determines that sufficient contextual information has been gathered to solve the task. The model converts its collected relevant code snippets into a formatted answer including their filenames, line, ranges, and a concise explanation. The formatted answer will then be parsed and guide the large generator model for answer generation.

3.2 Tool-based Memory Mechanism

To mitigate the high computational cost and performance degradation induced by excessively long conversations during multi-turn interactions, as depicted in Figure 1, we introduce a memory update mechanism that allows the model to invoke an additional tool for summarizing its exploration history and reducing conversation length. This tool dynamically manages a fixed-size context window by enabling selective retention and forgetting of information.

Specifically, when the length of the current context C_t exceeds a predefined threshold τ, a prompt is appended to the last observation o_t, instructing the explorer model to invoke the memory update tool. This tool requires the model to generate a summary of the exploration process and key findings based on the existing interaction history C_t and the current memory M_{prev}, which is initially empty. The generated summary serves as the updated memory M_{new} for subsequent interactions. Upon invocation, the conversation sequence is restructured: the sequence of interaction tuples t_i, a_i, o_i is replaced by the refreshed memory M_{new}. Formally, this transformation can be expressed as:

$$\langle p, M_{\text{prev}}, \{t_1, a_1, o_1\}, \ldots, \{t_k, a_k, o_k\}\rangle \quad \rightarrow \quad \langle p, M_{\text{new}}\rangle,$$

The explorer model continues the exploration process with the updated memory until it invokes the submit tool. This mechanism maintains the context window within a fixed size, preventing context overflow and preserving continuity in the exploration process.

3.3 Model Training

To further enhance the small explorer model's capabilities in repository exploration and memory updating, we propose a training methodology that combines cold-start training using trajectories distilled from large models with reinforcement learning.

First, in the cold start stage, we employ rejection sampling to collect high-quality training trajectories from the large model acting as an expert explorer agent. Specifically, for each sample in the training set, we sample 5 trajectory rollouts from the large model and retain only those that successfully lead to correct final answers. To improve training data quality, we further post-process these successful trajectories through the following filtering criteria: (1) we remove trajectories with invalid tool calls or malformed actions to ensure executable validity, (2) we filter out trajectories that are excessively long, exceeding a predefined maximum turn threshold, to avoid capturing inefficient exploration patterns, (3) we filter out trajectories that does not invoke the memory update tool when context C_t exceeds the threshold τ. Through fine-tuning on these trajectories, the small model learns to follow the effective exploration strategies exhibited by the large model, providing a

strong initialization before transitioning to reinforcement learning for further improvement.

Then, to avoid model just being overfitted on the expert trajectories obtained from large models and enhance its generalization, we employ reinforcement learning method GRPO [5] to empower the model to explore diverse paths in the interactive environment and train it on real-world feedback. We design trajectory-level and turn-level reward signals to optimize the model's context engineering strategies. The trajectory-level reward R_{traj} contains the *correctness reward* r_c which evaluates whether the curated context enables the generator LLM to produce the correct answer. Second, we introduce an *under-exploration penalty* p_u to discourage insufficient exploration when the final answer is incorrect. Specifically, let n denote the number of interaction turns and k represent a minimum turn threshold. When $r_c = 0$ and $n < k$, we apply a linear penalty $p_u = -\alpha(k - n)$, where $\alpha > 0$ is a hyperparameter controlling the penalty magnitude; otherwise, $p_u = 0$. At the turn level, we define R_{turn} to incorporate a penalty $p_m = -\beta$ (where $\beta > 0$) for each turn in which memory update should be invoked but is not executed by the model. Finally, following GRPO [5], the training objective is formulated as:

$$\mathcal{L} = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=1}^{T}\left(R_{\text{traj}} + R_{\text{turn},t} - \bar{R}\right)\nabla_\theta \log \pi_\theta(a_t|s_t)\right],$$

where π_θ denotes the explorer model's policy parameterized by θ, $\bar{R}$ represents the baseline reward computed as the group mean [5], and (s_t, a_t) denotes the state-action pair at timestep t.

4 PRELIMINARY EVALUATION

4.1 Experimental Setup

Datasets and Baselines. We evaluate ContextPilot on two representative repository-level code intelligence tasks: repository-level code generation and repository-level code QA. For code generation, we use DevEval [7] (cross-file portion) as evaluation benchmarks; for code QA, we employ LongCodeBench [10] (QA portion) for evaluation. Specifically, for each task, we randomly split the original data into training and testing set with a proportion of 8:2. We compare our method against several baselines: i) Direct Generation: generating answers without any contextual information; ii) RAG (BM-25): retrieval-augmented generation using BM-25 sparse retrieval to obtain relevant code snippet; iii) RAG (Embedding): retrieval-augmented generation using dense retrieval methods based on semantic embeddings; iv) Small Model Agent: an end-to-end agent using a small LLM as the base model; and v) Large Model Agent: an end-to-end agent using a large LLM as the base model. In baselines iv) and v), the models do not separate context engineering and answer generation into two distinct processes. Instead, they use view and search tools for exploration and directly submit the final answer once sufficient information has been collected.

Evaluation Metrics. We employ the same evaluation metrics used in the original datasets for consistency. For DevEval , we employ Pass@1 to measure functional correctness. For the LongCodeBench-QA, we use Accuracy as the percentage of correct responses over total multiple-choice questions. For efficiency evaluation, we follow previous work and use average cost as the metric to measure the overhead of different agent-based methods, where the token costs for different model sizes are referenced Together AI [1].

Table 1: Model performance on repo-level code generation.

Approach	Performance	Cost
Direct Generation	8.3	\$0.02
RAG (BM-25)	16.7	\$0.02
RAG (Embedding)	17.7	\$0.03
Small Model Agent	22.9	\$0.05
Large Model Agent	43.8	\$3.44
ContextPilot w/o training	36.5	\$0.10
ContextPilot	45.8	\$0.18

Table 2: Overall model performance on repo-level code QA.

Approach	Performance	Cost
Direct Generation	57.3	\$0.02
RAG (BM-25)	62.9	\$0.02
RAG (Embedding)	59.6	\$0.02
Small Model Agent	38.2	\$0.07
Large Model Agent	66.3	\$0.84
ContextPilot w/o training	62.9	\$0.13
ContextPilot	66.3	\$0.15

Implementation Details. We use Qwen-3-4B-Instruct-2507 as the small explorer model and use Qwen3-235B-A22B-Instruct-2507 as the large generator model. The value of τ, k, α and β are set to 6000, 10, 0.5 and 1, respectively. We conduct our experiments on a server with 4 A100-80G GPUs.

4.2 Experimental Results

The experimental results on both repository-level code generation and QA tasks demonstrate the effectiveness of ContextPilot in achieving strong performance with high computational efficiency.

Overall Performance. As shown in Tables 1 and 2, ContextPilot achieves comparable performance to Large Model Agent while requiring significantly lower cost. On code generation, ContextPilot achieves 45.8% Pass@1, outperforming Large Model Agent's 43.8% while reducing cost by 95% from \$3.44 to \$0.18. On code QA, ContextPilot matches Large Model Agent's 66.3% accuracy with only 18% of the cost at \$0.15 compared to \$0.84. These results demonstrate that the effectiveness of ContextPilot in code context engineering for repo-level tasks.

Benefits of Explorer-Generator Architecture. By comparing ContextPilot w/o training against Small Model Agent, we can find the benefits of our proposed architecture and memory mechanism. ContextPilot w/o training substantially outperforms Small Model Agent on both code generation, achieving 36.5% compared to 22.9%, with only modest cost increases. This demonstrates that our architecture effectively leverages the small explorer's navigation capabilities while delegating complex reasoning to the proficiency

model, yielding better results than both retrieval-based methods and small agent approaches.

Impact of Training. By further training the explorer model, ContextPilot achieves further improvements with 9.3% on code generation and 3.4% on code QA. This shows that our training approach improves the explorer's navigation strategies, ultimately enabling ContextPilot to match or exceed Large Model Agent performance while maintaining substantially lower computational overhead.

5 CONCLUSION AND FUTURE WORK

In this work, we propose a novel framework, ContextPilot, for effective and efficient context engineering in repository-level code tasks. ContextPilot integrates three core components: (1) a decoupled architecture where a lightweight explorer model discovers relevant context, while a large proficiency model generates the final output; (2) a tool-augmented memory mechanism to enable efficient context management; and (3) a reinforcement learning-driven training approach to optimize exploration strategies. Experiments on repository-level code generation and question answering benchmarks demonstrate that ContextPilot achieves competitive or even superior performance while significantly enhancing computational efficiency. For future work, we plan to refine the design of our method, conduct more experiments and in-depth result analysis, and investigate large-scale multi-task training across diverse repository-level tasks to strengthen the model's generalization across various scenarios.

REFERENCES

[1] Together AI. 2025. Together AI. https://www.together.ai/.

[2] Anthropic. 2025. Introducing Claude 4. *Anthropic News May 23, 2025* (2025). https://www.anthropic.com/news/claude-4

[3] ChatGPT. 2022. ChatGPT. https://chat.openai.com/.

[4] Cognition. 2025. SWE-grep. https://cognition.ai/blog/swe-grep.

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[6] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

[7] Jia Li, Ge Li, Yunfei Zhao, Yongmin Li, Huanyu Liu, Hao Zhu, Lecheng Wang, Kaibo Liu, Zheng Fang, Lanshen Wang, Jiazheng Ding, Xuanming Zhang, Yuqi Zhu, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, Yongbin Li, Bin Gu, and Mengfei Yang. 2024. DevEval: A Manually-Annotated Code Generation Benchmark Aligned with Real-World Code Repositories. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 3603–3614.

[8] Wei Liu, Ailun Yu, Daoguang Zan, Bo Shen, Wei Zhang, Haiyan Zhao, Zhi Jin, and Qianxiang Wang. 2024. GraphCoder: Enhancing Repository-Level Code Completion via Coarse-to-fine Retrieval Based on Code Context Graph. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, Vladimir Filkov, Baishakhi Ray, and Minghui Zhou (Eds.). ACM, 570–581.

[9] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023).

[10] Stefano Rando, Luca Romani, Alessio Sampieri, Yuta Kyuragi, Luca Franco, Fabio Galasso, Tatsunori Hashimoto, and John Yang. 2025. LongCodeBench: Evaluating Coding LLMs at 1M Context Windows. *CoRR* abs/2505.07897 (2025).

[11] Chaozheng Wang, Zezhou Yang, Shuzheng Gao, Cuiyun Gao, Ting Peng, Hailiang Huang, Yuetang Deng, and Michael R. Lyu. 2025. RAG or Fine-tuning? A Comparative Study on LCMs-based Code Completion in Industry. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, FSE Companion 2025, Clarion Hotel Trondheim, Trondheim, Norway, June 23-28, 2025*, Leonardo Montecchi, Jingyue Li, Denys Poshyvanyk, and Dongmei Zhang (Eds.). ACM, 93–104.

[12] Yanlin Wang, Yanli Wang, Daya Guo, Jiachi Chen, Ruikai Zhang, Yuchi Ma, and Zibin Zheng. 2025. RLCoder: Reinforcement Learning for Repository-Level Code Completion. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 1140–1152.

[13] Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. 2024. Repoformer: Selective Retrieval for Repository-Level Code Completion. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

[14] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.).

[15] 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, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 2471–2484.

[16] Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024. CodeAgent: Enhancing Code Generation with Tool-Integrated Agent Systems for Real-World Repo-level Coding Challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 13643–13658.