Incorporating Chain-of-Context in Self-planning Enhances Interactive Code Generation from Natural Language

Anonymous ACL submission

Abstract

Recent progress in research on Large Language Models have significantly advanced their ability to handle coding tasks. Given an intent in natural language and relevant context, they can solve a wide-range of problems involving code generation. A lot of approaches built for using LLMs to solve coding tasks, apply self-planning, which includes a planning phase to guide the code generation step. But real-world problems involving code generation usually requires working on existing code and doing modifications to it complete the task at hand. Additionally, the natural language intent specifying such tasks don't usually include all the details required to solve it. We propose a method, Chain-Of-Context, that recursively performs self-planning and context curation to gather all relevant information to solve the task. Our key motivation is that LLMs, given an under-specified intent, can recursively decompose it into several specific sub-tasks and curate relevant contexts required at each step to solve the problem.

1 Introduction

Large Language Models (LLMs) have significantly improved in generating code snippets from user task descriptions. While state-of-the-art models achieve high performance on benchmarks focused on snippet-level generation, real-world coding often involves under-specified task descriptions and often requires changes in multiple files in a repository. Hence, focus of evaluation benchmarks have shifted towards repository-level tasks that better represent practical software development, requiring interaction with entire codebases rather than isolated snippet generation. SWE-Bench (Jimenez et al., 2024) is the most prominent benchmark for evaluating LLMs on such tasks, mirroring fundamental software engineering development. Each SWE-Bench problem comprises of an issue from an open-source Python Github repository and a corresponding commit ID defining the repository state at the time the issue was reported. The task is to resolve the issue via repository code changes through a git patch, replicating an environment where codebase understanding and navigation are of fundamental importance.

Research addressing real-world software development problems, evaluated via benchmarks like SWE-Bench, has largely focused on developing complex agent scaffolds (Wang et al., 2024; Yang et al., 2024; Xia et al., 2024; Aggarwal et al., 2025). These agents typically perform code searching, response sampling, and test execution. Such scaffolds have progressively improved, increasing solve rate on SWE-Bench. A critical component within these systems is fetching relevant code context. This work focuses specifically on enhancing this context-gathering process to effectively disambiguate under-specified task descriptions and generate code with higher accuracy. We propose a recursive method for self-planned context curation, ensuring comprehensive information acquisition for problem understanding and resolution. The main intuition is that context relevant to the current sub-task enhances understanding of the problem and the required solution steps. Furthermore, particularly when resolving codebase issues where context is essential for determining necessary code modifications, this approach can refine observed bug descriptions into root causes by planning next sub-task based on previous sub-tasks and retrieved relevant context, thereby helping with targeted accurate fixes.

We propose Chain-Of-Context in Self-Planning, which focuses on fetching relevant context for each planned sub-task to gather all information to finally generate the code solution. We evaluate the method on SWE-Bench-Verified and SWE-Bench-Lite splits. It significantly outperforms self-planning with Retrieval-Augmented Generation (RAG) (Lewis et al., 2020), while only marginally

better than targeted context retrieval from the codebase conditioned on task description while incurring significantly higher computational cost per instance.

2 Proposed Method: Chain-of-Context

The proposed method Chain-of-Context in Self-Planning involves the following key steps:

1. Given the user intent, codebase structure and all the relevant context (in the first step there is no available context), the model is prompted to output the sub-task it wants to execute.

2. Retrieve context from the codebase for the sub-task generated by the model.

3. Conditioned on the user intent, all the relevant retrieved context and the sub-task description, the model is prompted to determine if the retrieved context is relevant to the sub-task and the problem in the user intent. If the model determines it to be relevant, the retrieved context is added to the list of previously retrieved relevant context.

4. Recursively perform steps 1 to 3 until the model determines it has retrieved all the relevant context required and decides to implement the code solution in its next sub-task.

5. Given the user intent, and all the relevant context, the model is prompted to provide the code implementation for the solution to the problem.

Specific implementation details and design choices adapted according to the task type are elaborated in the subsequent section.

3 Experimentation Setup

3.1 Models

For all experiments, we utilize GPT-4o (Hurst et al., 2024) and O3-mini (OpenAI, 2025), two models commonly employed by other SWE-Bench evaluation scaffolds, enabling comparison with existing methods.

3.2 Datasets

The main requirement for evaluating our method is the use of coding tasks complex enough to necessitate decomposition into multiple sub-tasks and it's supporting codebase exceeds the model's single-prompt maximum tokens limit. We experiment on SWE-Bench as it meets requirements, which reflect typical real-world coding challenges.

SWE-Bench consists of 2,294 real-world coding problems, where each sample is an issue description from a Github repository and a commit ID defining the repository state at the time the issue was reported. The task is to generate a git patch that resolves the issue. Our experiments utilize two subsets: SWE-Bench-Lite and SWE-Bench-Verified. SWE-Bench-Lite consists of 300 samples selected for lower complexity, involving only a single-file change to solve the issue. SWE-Bench-Verified includes 500 samples, human-verified to ensure the corresponding issue description provides sufficient, unambiguous detail for a system to solve the problem using the provided codebase, unlike the full dataset. Consequently, SWE-Bench-Verified is considered to have a theoretical maximum solve rate of 100%.

3.3 Chain-of-Context Agentic Framework

We implement an agent-based framework applying the proposed Chain-of-Context method, enabling context retrieval from the codebase and generation of a valid git patch that resolves the given issue. The core architecture of this framework, integrating Chain-of-Context, is depicted in Figure 1 and implemented as follows:

1. In the ANALYZE stage, the agent (LLM) is prompted with the user's intent (issue description), a structured list of all Python files (formatted as modules), and previously retrieved relevant code snippets. It is prompted to produce a sub-task, either requesting further code context or providing a root cause analysis (RCA) of the issue.

2. If the agent requests additional context, the query is handled in the RETRIEVE stage by a retriever component built upon abstract syntax tree (AST) parsing. This component indexes the repository's codebase using AST parsing to efficiently retrieve specific classes, functions, or entire files matching the agent's request. The retriever returns the closest matching code snippet for the agent's query.

3. In the VERIFY stage, the retrieved code context undergoes a verification step, where the LLM evaluates the retrieved code alongside

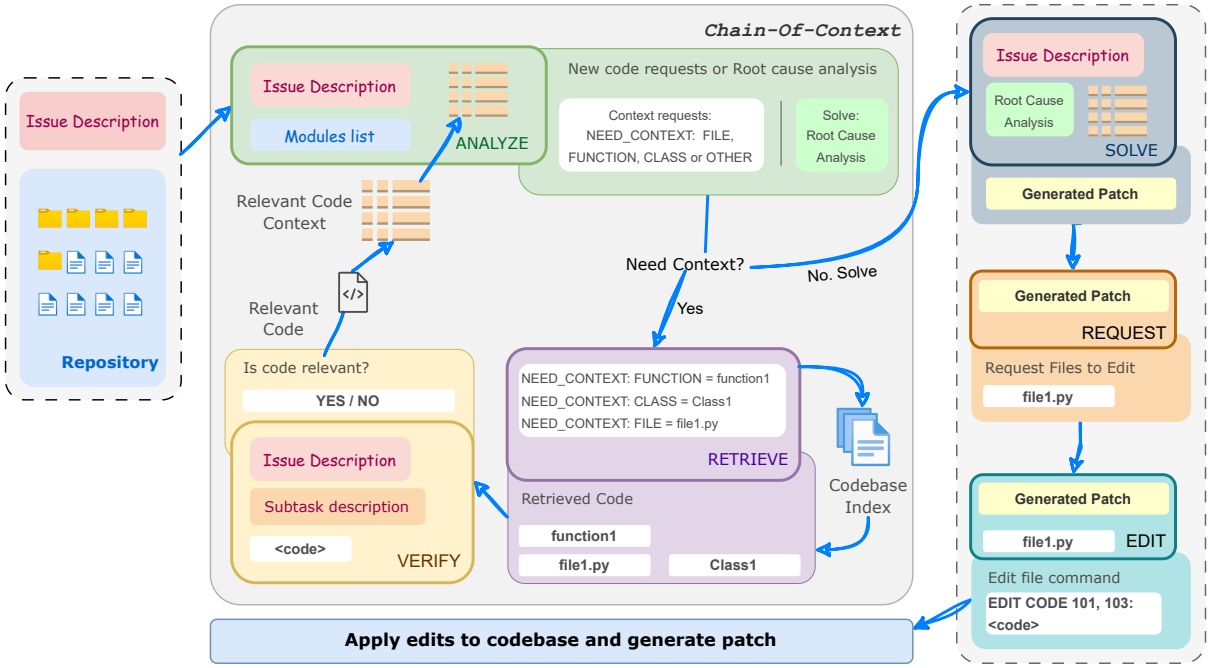

Figure 1: Chain-Of-Context framework. It performs the core context curation as part of a recursive process between ANALYZE, RETRIEVE and VERIFY stages until it determines it has gather all relevant context to generate a solution. The control then flows to the Patch Generation block which includes the SOLVE, REQUEST and EDIT stages to generate a valid patch.

the sub-task description to check its relevance. If it is found to be relevant, the code context is retained and added to the existing list of relevant code; otherwise, it is discarded.

4. The agent recursively cycles through the ANALYZE, RETRIEVE and, VERIFY stages to curate the relevant code required to provide its complete root cause analysis. Once, it determines that it has enough information to debug the issue, it provides its root cause analysis (RCA) to solve the issue in its next sub-task, and the control is passed to the SOLVE stage.

5. In the SOLVE stage, the LLM, conditioned on the issue description, all the retrieved relevant code for sub-tasks, and it's RCA, generates a code solution in unified diff format.

6. Patch Generation: To ensure that the diff is valid and applicable as a git patch, we employ a file-editor component. The file editor component comprises REQUEST and EDIT stages. In the REQUEST stage, the LLM requests for files to edit conditioned on the solution in the unified diff format generated as part of the SOLVE stage. In the EDIT stage, the LLM conditioned on the file content and the solution, sequentially edits requested files, explicitly specifying line numbers and corresponding new code snippets for the edit. These edits are applied to a locally cloned repository checked out at the sample's base commit, resulting in a correctly formatted git patch file. Post-generation, the applied changes are reverted. This patch generation approach achieves near-perfect accuracy, thus concentrating research efforts on enhancing the model's coding and reasoning capabilities.

3.4 Baselines

We compare our proposed method with two baselines to estimate the efficacy of our method while isolating Chain-Of-Context to accurately determine its impact.

Self-Planning with Retrieval Augmented Generation (RAG) The first baseline combines self-planning (Jiang et al., 2024) with Retrieval Augmented Generation (RAG) to solve the issue. The codebase is chunked and stored as embeddings which is queried using the issue description. We use text-embedding-3-small model with code aware chunking to generate embeddings for the codebase. The model is prompted to solve the is-

SWE-Bench-Verified					
Method	**Model**	**% Resolved**	**Avg. Cost (Emb. cost)**	**% Correct Location**	
				File	**Line**
Self-Planning with RAG	GPT-4o	3.40	$0.07 ($0.05)	52.60	12.00
	O3-mini	1.20	$0.06 ($0.05)	55.40	12.00
Self-Planning with ICR	GPT-4o	11.40	$0.16	60.80	25.80
	O3-mini	12.60	$0.10	63.80	33.80
Chain-Of-Context	GPT-4o	14.00	$0.28	64.00	27.00
	O3-mini	14.40	$0.16	68.40	36.80
CodeStory Midwit Agent	_	62.20	_	80.00	45.71
OpenHands + 4x Scaled	_	60.80	_	78.80	35.80
Agentless-1.5	GPT-4o	38.80	_	66.00	31.80
Agentless Lite	O3-mini	42.40	_	70.00	38.57
SWE-Bench-Lite					
Method	**Model**	**% Resolved**	**Avg. Cost (Emb. cost)**	**% Correct Location**	
				File	**Line**
Self-Planning with RAG	GPT-4o	3.67	$0.07 ($0.05)	68.67	18.33
	O3-mini	1.33	$0.06 ($0.05)	69.33	18.67
Self-Planning with ICR	GPT-4o	9.00	$0.16	63.67	28.00
	O3-mini	10.67	$0.09	67.33	32.33
Chain-Of-Context	GPT-4o	9.33	0.26	65.33	24.67
	O3-mini	10.67	$0.15	70.33	35.67
DARS Agent	_	47.00	_	83.00	41.67
Kodu-v1	_	44.67	_	87.06	37.06
Agentless-1.5	GPT-4o	32.00	_	69.67	31.67
Agentless Lite	O3-mini	32.33	_	76.19	39.11

Table 1: Results on SWE-Bench-Verified and SWE-Bench-Lite using three methods: Self-Planning with RAG, Self-Planning with ICR, and Chain-of-Context in Self-Planning. Additionally, we report scores for the top open-source scaffolds from the SWE-Bench leaderboard, including CodeStory Midwit Agent (Codestory, 2024), OpenHands + 4x Scaled (Wang et al., 2024), DARS Agent (Aggarwal et al., 2025), Kodu-v1 (Kodu-Ai), Agentless-1.5 (Xia et al., 2024), and Agentless Lite (Xia et al., 2024).

sue using self-planning conditioned on the issue description and the documents fetched during retrieval.

Self-Planning with Indexed Context Retrieval (ICR) The second baseline uses the framework depicted in Figure 1, but excludes the Chain-of-Context component. In this setup, the model receives the issue description, self-plans and performs a single context request, and subsequently generates the solution conditioned on issue description, its generated RCA and the retrieved code context. The solution generation stage follows the

SOLVE, REQUEST, and EDIT phases detailed in 1. This baseline enables us to precisely isolate and measure the performance impact of the Chain-of-Context method in our proposed approach.

3.5 Metrics

We report the percentage of solved samples and the computational cost per sample for all three methods on the SWE-Bench-Verified and SWE-Bench-Lite datasets. For the first baseline, the reported cost additionally includes the embedding generation for each codebase.

Additionally, we compare our results against the top two open-source scaffolds, as well as the best-performing scaffolds using GPT-4o and o3-mini on both SWE-Bench-Verified and SWE-Bench-Lite, as reported on the SWE-Bench leaderboard. These scaffolds leverage multiple solution samplings and validate outputs via test case execution—techniques not included in our approach. To ensure a fair comparison, we focus on the % Correct Location metric as defined in Agentless (Xia et al., 2024), which evaluates the ability of Chain-of-Context to accurately localize the issue. A solution is considered correct if it covers all edit locations present in the ground truth patch. We report this metric at two levels of granularity: file and line.

4 Results

The experimentation results are presented in Table 1. On the SWE-Bench-Verified dataset, our Chain-of-Context approach significantly outperforms the Self-Planning with RAG baseline but shows only a marginal improvement over Self-Planning with ICR, while incurring notably higher computational costs due to recursive context retrieval. This trend is consistent across both models (GPT-4o and O3-mini), as well as on the SWE-Bench-Lite dataset. Finally, we would like to highlight that Chain-Of-Context performs significantly worse than the best open source scaffolds that involve additionally tooling to suppor the solution generation.

To enable a fair comparison with the best open-source scaffolds, we use the % Correct Location metric, as our method does not include multiple response sampling or test execution tooling which are generally used in the reported scaffolds. While we acknowledge that an issue can be resolved at multiple locations, we observe a correlation between this metric and the solved percentage, as also noted in Agentless (Xia et al., 2024). Chain-of-Context with O3-mini approaches the performance of open-source scaffolds in localizing issues at both file and line granularities across both datasets, but shows only minor gains over Self-Planning with ICR, despite higher computational costs. Self-Planning with RAG generally struggles with accurate localization.

5 Conclusion

In conclusion, we implement Chain-of-Context into Self-Planning to address complex coding tasks with under-specified intent by retrieving targeted context for each planned sub-task and generating solutions conditioned on the aggregated information. The core intuition is to curate context at the sub-task level to collect all relevant details necessary to resolve the issue. Our results show that while Chain-of-Context significantly outperforms RAG, it offers only marginal improvements over single-shot context retrieval without creating sub-tasks. Additionally, we find that Chain-of-Context improves localization of code changes; however, solving more complex tasks may require additional software development tooling to support accurate solution generation.

References

Vaibhav Aggarwal, Ojasv Kamal, Abhinav Japesh, Zhijing Jin, and Bernhard Schölkopf. 2025. Dars: Dynamic action re-sampling to enhance coding agent performance by adaptive tree traversal.

Codestory. 2024. Sota on swebench-verified: (re)learning the bitter lesson.

OpenAI Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mkadry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alexander Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alexandre Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, B. Ghorbani, Ben Leimberger, Ben Rossen, Benjamin Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen

Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll L. Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Chris Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mély, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Phong Duc Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Hai-Biao Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Pondé de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian D. Kivlichan, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, İbrahim Cihangir Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub W. Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Ryan Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quiñonero Candela, Joe Beutler, Joe Landers, Joel Parish, Jo hannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Joshua Gross, Josh Kaplan, Josh Snyder, Josh Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Ouyang Long, Louis Feuvrier, Lu Zhang, Lukasz Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Made laine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Ma teusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Ali Yatbaz, Mengxue Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Mina Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nickolas Stathas, Nick Turley, Nikolas A. Tezak, Niko Felix, Nithanth Kudige, Nitish Shirish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Phil Tillet, Prafulla Dhariwal, Qim ing Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Raphael Gontijo Lopes, Raul Puri, Reah Miyara, Reimar H. Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Ramilevich Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal A. Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. 2024. Gpt-4o system card. *ArXiv*, abs/2410.21276.

Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. 2024. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–30.

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*.

Kodu-Ai. Kodu-ai/claude-coder.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020.

Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

OpenAI. 2025. Openai o3-mini.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. 2024. OpenHands: An Open Platform for AI Software Developers as Generalist Agents.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying llm-based software engineering agents.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.