THE LIMITS OF LONG-CONTEXT REASONING IN AUTOMATED BUG FIXING

Ravi Raju,* **Mengmeng Ji,*** **Shubhangi Upasani, Bo Li & Urmish Thakker**
SambaNova Systems
San Jose, CA 95134, USA
{ravi.raju,mengmeng.ji}@sambanovasystems.edu

ABSTRACT

Rapidly increasing context lengths have led to the assumption that large language models (LLMs) can directly reason over entire codebases. Concurrently, recent advances in LLMs have enabled strong performance on software engineering benchmarks, particularly when paired with agentic workflows. In this work, we systematically evaluate whether current LLMs can reliably perform long-context code debugging and patch generation. Using SWE-bench Verified as a controlled experimental setting, we first evaluate state-of-the-art models within an agentic harness (mini-SWE-agent), where performance improves substantially: GPT-5-nano achieves up to a 31% resolve rate on 100 samples, and open-source models such as Deepseek-R1-0528 obtain competitive results. However, token-level analysis shows that successful agentic trajectories typically remain under 20k-30k tokens, and that longer accumulated contexts correlate with lower success rates, indicating that agentic success primarily arises from task decomposition into short-context steps rather than effective long-context reasoning. To directly test long-context capability, we construct a data pipeline where we artificially inflate the context length of the input by placing the relevant files into the context (ensuring perfect retrieval recall); we then study single-shot patch generation under genuinely long contexts (64k tokens). Despite this setup, performance degrades sharply: Qwen3-Coder-30B-A3B achieves only a 7% resolve rate at 64k context, while GPT-5-nano solves none of the tasks. Qualitative analysis reveals systematic failure modes, including hallucinated diffs, incorrect file targets, and malformed patch headers. Overall, our findings highlight a significant gap between nominal context length and usable context capacity in current LLMs, and suggest that existing agentic coding benchmarks do not meaningfully evaluate long-context reasoning.

1 INTRODUCTION

Supported context lengths in large language models (LLMs) have expanded dramatically, fostering the expectation that models can directly reason over entire codebases in a single pass (DeepSeek-AI, 2025; Team, 2025). Meanwhile, recent advances in LLMs have led to strong performance on software engineering tasks such as code generation and debugging (Jimenez et al., 2024; Liu et al., 2023; Li et al., 2025). Whether current LLMs can effectively use such long contexts in realistic programming scenarios, however, remains unclear.

In this work, we study the limits of long-context code reasoning by evaluating LLMs on repository-scale debugging tasks using SWE-bench Verified as a controlled experimental setting. SWE-bench Verified is a curated subset of the SWE-bench benchmark consisting of real GitHub issues paired with code repositories and human-validated patches, where each problem has been carefully verified to ensure the issue description, tests, and environment reliably reproduce the bug. We begin by examining conventional ways to evaluate LLMs' agentic coding capabilities through open-source agent scaffolding such as SWE-agent and OpenHands (Wang et al., 2025a). We test current SOTA models like GPT-5-nano, Deepseek R1-0528 and Qwen3-32B, for their accuracy on SWE-bench

*Equal contribution.

Verified and how many tokens, on average, are produced in order to complete the aforementioned task.

Reasoning models like Deepseek R1-0528 and Qwen3-32B are known to produce a significant amount of output tokens for a given task. Therefore, on first glance, one may assume that a software engineering task like SWE-bench, which may require multiple invocations of the agentic system, to solve will invariably end up being long context. However, we find that through token length analysis of agentic trajectories that they tend to be under 20k-30k tokens, clearly not long context by current standards. This demonstrates that strong agentic performance should not be interpreted as evidence of long-context capability.

In order to isolate the impact of the LLM's coding ability and to isolate long context coding performance, we examine single-shot patch generation under long contexts (64k), where all files required to construct the ground-truth patch are explicitly provided, eliminating retrieval as a confounding factor. Under this setup, state-of-the-art models, such as Qwen3-Coder-30B-A3B and GPT-5-nano, perform poorly: at 64k context, resolve rates remain in the single digits or zero (much lower than the agentic resolve rates). These failures include manifesting as hallucinated edits which result in malformed diffs that cannot be applied as well as referencing files which don't exist in the context.

These findings indicate that increased context length alone does not translate into reliable long-context reasoning. In contrast, the same models achieve substantially higher resolve rates when evaluated in agentic frameworks (*mini-swe-agent*) that decompose the task into multiple short-context steps. While effective, such workflows largely bypass long-context reasoning rather than exercising it.

Our contributions are as follows:

1. We provide token-length analysis of reasoning traces to demonstrate that common agentic coding evaluations do not meaningfully exercise long-context reasoning.
2. We present qualitative observations of SOTA LLMs failure modes in long-context, single-shot code patch generation.

Together, these findings highlight a substantial gap between nominal context length and usable context capacity in current LLMs for software engineering, and motivate the need for models that explicitly target long-context reasoning rather than assuming it emerges implicitly from agentic interaction.

2 Comparative Analysis of Agentic and Long-Context Reasoning

We begin by evaluating LLMs on agentic workflows based on *mini-swe-agent*, analyzing how task decomposition alters effective context usage Yang et al. (2024). We then contrast these results with single-shot patch generation under genuinely long contexts using retrieval-augmented inputs with perfect recall, exposing the failure modes of direct long-context reasoning. We provide more detailed descriptions of the benchmark in the Appendix A.1.

2.1 Evaluating Usable Context in Agentic Frameworks

We evaluate SWE-bench in an agentic workflow setting using the *mini-SWE-agent* framework, the reference agent released by the SWE-bench authors (Yang et al., 2024). Unlike SWE-agent, which leverages an ACI (Agent-Computer Interface), mini-SWE-agent is a bash-only command-line workflow that provides a lightweight sandbox for experimenting with different models as shown in Figure 3 (in Appendix). It produces a linear history in which each agent step is appended directly to the message stream—making it especially useful for debugging, fine-tuning, and measuring token usage and conversation length. This design aligns well with our goal of assessing SWE-bench's strengths as an agentic benchmark while also probing its potential as a long-context benchmark.

SWE-bench has multiple different version: SWE-bench, SWE-bench Lite and SWE-bench Verified. SWE-bench Verified is the best out of all these options as it removes unsolvable or irrelevant issues and categorizing problems by difficulty (Chowdhury et al., 2024). As a sanity check for our evaluation infrastructure, we tested GPT-5-nano and obtained a 31% resolve rate on 100 samples

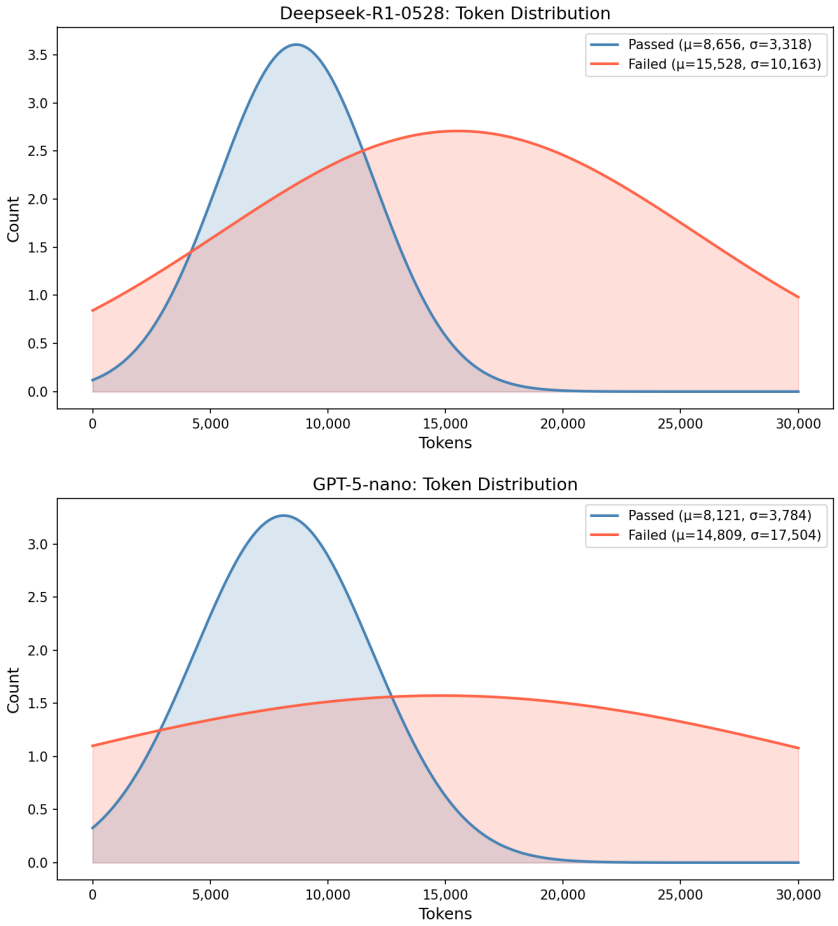

Figure 1: We show the token distributions of the model trajectories for Deepseek-R1-0528 and GPT-5-nano in terms of mean and variance for passed/failed category. Deepseek-R1-0528 has 30 samples and 69 samples in the passed and fail category, respectively, whereas GPT-5-nano has 31 samples and 69 samples in the passed and fail category. The distribution for each model shows that samples which the model fail to answer tend to consume more tokens. These figure illustrate that the average conversation/trajectory is still under 20-30k and can't be considered a long context interaction.

from SWE-bench Verified using the *sb-cli* tool, the official SWE-bench command-line evaluation tool. GPT-5 nano's results on the open-source leaderboard on the full dataset is 34.8%, which both derisks our evaluation infrastructure and justifies our choice of using a subsample of the dataset for evaluation (Chowdhury et al., 2024). With our harness's fidelity verified, we evaluate 2 SOTA open-source models, DeepSeek R1-0528 and Qwen3-32B using the SambaNova Cloud API (DeepSeek-AI, 2025; Team, 2025). On 100 samples from SWE-bench Verified, DeepSeek R1-0528 achieved a 30.3% resolve rate, while Qwen3-32B reached 15.2%.

While these experiments confirm that SWE-bench is well-suited for agentic workflows, they also raise the question: *can it still serve as a long-context benchmark*? A key challenge lies in measuring and controlling context length in an agentic setting. Context length here depends on the number of dialogue rounds and accumulated tokens. In Figure 1, we show the trajectory token distributions for both Deepseek-R1-0528 and GPT-5-nano (please refer to the Appendix for Qwen3-32B's token distribution) full conversations generally stayed under 20k-30k tokens. Within this range, successful solutions tended to use shorter contexts, while samples with longer contexts tended to have lower resolve rates. It is unclear whether or not if longer contexts have worse resolve rates due to the fact that more context could confuse/complicate reasoning vs. more challenging issues require reasoning and are harder to resolve; disentangling these two factors is a good direction for future work. In

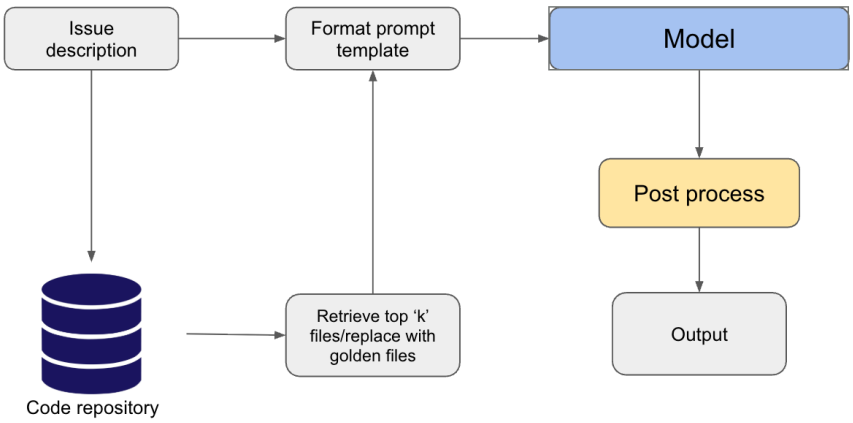

Figure 2: Our long-context gold patch generation pipeline. By retrieving/injecting all the files present in the ground truth gold patch, we isolate the LLM's patch generation ability.

totality, these findings indicate that SWE-bench is most suitable as an agentic workflow benchmark, but is less effective for evaluating long-context coding capabilities.

2.2 Stress-Testing Direct Reasoning in 64k Context Windows

In order to test solely the long context coding ability of models, we need to create long context evaluation with the existing tools. However, unlike SWE-bench Lite, SWE-bench Verified provides no oracle sources or files. As a result, we explored RAG techniques to retrieve long context(s) for model generation.

```
--- a/lib/matplotlib/offsetbox.py
+++ b/lib/matplotlib/offsetbox.py
@@ -159,26 +159,37 @@ def _get_aligned_offsets(hd_list, height, align
    ="baseline"):
- if height is None:
- height = max(h for h, d in hd_list)
- _api.check_in_list(
- ["baseline", "left", "top", "right", "bottom", "center"], align=
    align)
-
- if align == "baseline":
- height_descent = max(h - d for h, d in hd_list)
- descent = max(d for h, d in hd_list)
- height = height_descent + descent
- offsets = [0. for h, d in hd_list]
- elif align in ["left", "top"]:
- descent = 0.
- offsets = [d for h, d in hd_list]
- elif align in ["right", "bottom"]:
- descent = 0.
- offsets = [height - h + d for h, d in hd_list]
- elif align == "center":
```

Listing 1: A misaligned patch generated by GPT-5 nano. One of the errors can be seen in the chunk header, where the line numbers from the first line clearly exceed the patch length.

Based on our literature survey, major retrieval methods are either based on sparse retrieval methods like BM25, or more complex agentic systems, like meta-RAG, Code-RAG bench, or a hybrid variant like CodeMonkeys (Tawosi et al., 2025; Wang et al., 2025b; Ehrlich et al., 2025; Robertson & Zaragoza, 2009). Given the high cost of agent-based retrieval, we opted for a simpler yet effective strategy: use BM25 to rank code chunks and then inject the "gold" patches to guarantee high recall. With this approach, we curated a smaller dataset from SWE-bench Verified (using the same dataset

as above), containing 100 samples with 100% recall (meaning all the required files are within the context), one with 64k token context length. We illustrate our pipeline in Figure 2.

Using this long-context dataset, we evaluated two models: GPT-5-nano and Qwen3-Coder-30B-A3B, with *sb-cli* Team (2025); Singh et al. (2025). We chose to evaluate these models instead of Deepseek-R1-05-28 or Qwen3-32B since at the time of writing, we didn't have access to the compute resources which would allow us to run inference at the long sequence length we desired.

On our 64k setting, the results were poor: Qwen3-Coder-30B-A3B achieved only a 7% resolve rate, while GPT-5-nano solved none of the tasks. We conducted qualitative analysis on the model's outputs and revealed common failure modes. Many generated patches contained hallucinated information: some had incorrect line numbers in the diff header as shown in Listing 1, while others targeted files that did not exist in the repository. After manually verifying that our post-processing pipeline was not the source of these errors, we conclude that single-shot prompting under genuinely long contexts exceeds the effective reasoning capacity of current LLMs for repository-scale debugging. While these models nominally support long context windows, our results indicate that, when faced with complex, multi-file code changes, they frequently fail to reliably synthesize and act on dispersed information, leading to hallucinated or malformed patches. These findings may suggest that using specialized harnesses are critical to achieving competitive performance on SWE tasks, as seen in Erdil (2025). As a result, SWE-bench is not suitable as a single-shot long-context standalone benchmark, since failures persist even under perfect retrieval and instead reflect practical limits in current LLMs' ability to operate at this scale.

3 Conclusion

Although SWE-bench is valuable for testing coding agents, it is not suitable for long-context evaluation. Our results challenge the assumption that repeated calls in agentic settings naturally lead to long-context use. In RAG/oracle experiments, models failed to solve long-context tasks in a single shot. Recently, LongCodeBench introduced LongSWE-bench, a similar benchmark to our RAG/oracle setting, showing that open-source models achieved only single-digit solve rates, and the best closed-source model (Gemini 2.5 Pro) solved just 22% Rando et al. (2025). In essence, our observations are two-fold: 1) we need more long-context focused versions of SWE-bench to test long horizon coding tasks and 2) the current model capabilities are not sufficient to perform well on long-context (single-shot) code debugging. Taken together, these results underscore the need for targeted advances in software engineering agents that explicitly address long-context reasoning, rather than assuming such capabilities will emerge naturally from agentic interaction alone.

4 Ethics Statement

This work focuses on evaluating the long-context capabilities of existing large language models using established, publicly available benchmarks and evaluation frameworks. All experiments are conducted on open-source datasets (SWE-bench and SWE-bench Verified) derived from publicly accessible GitHub repositories, and we do not introduce new data collection involving human subjects or sensitive personal information.

Our study does not involve training or deploying models in real-world software development environments. Instead, it analyzes model behavior in controlled offline settings to better understand the limitations of long-context reasoning and agentic workflows. While our findings highlight failure modes such as hallucinated or malformed patches, these results are presented for diagnostic and benchmarking purposes and are not intended to encourage automated code deployment without human oversight.

We believe that clarifying the limits of current LLM capabilities, particularly in safety-critical tasks such as code modification, is a positive contribution that can help prevent misuse and overreliance on automated systems. No foreseeable ethical risks arise beyond those already associated with the use of large language models for code generation, which are mitigated by the evaluation-only nature of this work.

REFERENCES

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. Codemonkeys: Scaling test-time compute for software engineering, 2025. URL https://arxiv.org/abs/2501.14723.

Ege Erdil. What skills does swe-bench verified evaluate?, June 2025. URL https://epoch.ai/blog/what-skills-does-swe-bench-verified-evaluate. Accessed: 2026-03-05.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Jia Li, Xuyuan Guo, Lei Li, Kechi Zhang, Ge Li, Jia Li, Zhengwei Tao, Fang Liu, Chongyang Tao, Yuqi Zhu, and Zhi Jin. Longcodeu: Benchmarking long-context language models on long code understanding, 2025. URL https://arxiv.org/abs/2503.04359.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems, 2023. URL https://arxiv.org/abs/2306.03091.

Stefano Rando, Luca Romani, Alessio Sampieri, Luca Franco, John Yang, Yuta Kyuragi, Fabio Galasso, and Tatsunori Hashimoto. Longcodebench: Evaluating coding llms at 1m context windows, 2025. URL https://arxiv.org/abs/2505.07897.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, Alex Makelov, Alex Neitz, Alex Wei, Alexandra Barr, Alexandre Kirchmeyer, Alexey Ivanov, Alexi Christakis, Alistair Gillespie, Allison Tam, Ally Bennett, Alvin Wan, Alyssa Huang, Amy McDonald Sandjideh, Amy Yang, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrei Gheorghe, Andres Garcia Garcia, Andrew Braunstein, Andrew Liu, Andrew Schmidt, Andrey Mereskin, Andrey Mishchenko, Andy Applebaum, Andy Rogerson, Ann Rajan, Annie Wei, Anoop Kotha, Anubha Srivastava, Anushree Agrawal, Arun Vijayvergiya, Ashley Tyra, Ashvin Nair, Avi Nayak, Ben Eggers, Bessie Ji, Beth Hoover, Bill Chen, Blair Chen, Boaz Barak, Borys Minaiev, Botao Hao, Bowen Baker, Brad Lightcap, Brandon McKinzie, Brandon Wang, Brendan Quinn, Brian Fioca, Brian Hsu, Brian Yang, Brian Yu, Brian Zhang, Brittany Brenner, Callie Riggins Zetino, Cameron Raymond, Camillo Lugaresi, Carolina Paz, Cary Hudson, Cedric Whitney, Chak Li, Charles Chen, Charlotte Cole, Chelsea Voss, Chen Ding, Chen Shen, Chengdu Huang, Chris Colby, Chris Hallacy, Chris Koch, Chris Lu, Christina Kaplan, Christina Kim, CJ Minott-Henriques, Cliff Frey, Cody Yu, Coley Czarnecki, Colin Reid, Colin Wei, Cory Decareaux, Cristina Scheau, Cyril Zhang, Cyrus Forbes, Da Tang, Dakota Goldberg, Dan Roberts, Dana Palmie, Daniel Kappler, Daniel Levine, Daniel Wright, Dave Leo, David Lin, David Robinson, Declan Grabb, Derek Chen, Derek Lim, Derek Salama, Dibya Bhattacharjee, Dimitris Tsipras, Dinghua Li, Dingli Yu, DJ Strouse, Drew Williams, Dylan Hunn, Ed Bayes, Edwin Arbus, Ekin Akyurek, Elaine Ya Le, Elana Widmann, Eli Yani, Elizabeth Proehl, Enis Sert, Enoch Cheung, Eri Schwartz, Eric Han, Eric Jiang, Eric Mitchell, Eric Sigler, Eric Wallace, Erik Ritter, Erin Kavanaugh, Evan Mays, Evgenii Nikishin, Fangyuan Li, Felipe Petroski Such, Filipe de Avila Belbute Peres, Filippo Raso, Florent Bekerman, Foivos Tsimpourlas, Fotis Chantzis, Francis Song, Francis Zhang, Gaby Raila, Garrett McGrath, Gary Briggs, Gary Yang,

Giambattista Parascandolo, Gildas Chabot, Grace Kim, Grace Zhao, Gregory Valiant, Guillaume Leclerc, Hadi Salman, Hanson Wang, Hao Sheng, Haoming Jiang, Haoyu Wang, Haozhun Jin, Harshit Sikchi, Heather Schmidt, Henry Aspegren, Honglin Chen, Huida Qiu, Hunter Lightman, Ian Covert, Ian Kivlichan, Ian Silber, Ian Sohl, Ibrahim Hammoud, Ignasi Clavera, Ikai Lan, Ilge Akkaya, Ilya Kostrikov, Irina Kofman, Isak Etinger, Ishaan Singal, Jackie Hehir, Jacob Huh, Jacqueline Pan, Jake Wilczynski, Jakub Pachocki, James Lee, James Quinn, Jamie Kiros, Janvi Kalra, Jasmyn Samaroo, Jason Wang, Jason Wolfe, Jay Chen, Jay Wang, Jean Harb, Jeffrey Han, Jeffrey Wang, Jennifer Zhao, Jeremy Chen, Jerene Yang, Jerry Tworek, Jesse Chand, Jessica Landon, Jessica Liang, Ji Lin, Jiancheng Liu, Jianfeng Wang, Jie Tang, Jihan Yin, Joanne Jang, Joel Morris, Joey Flynn, Johannes Ferstad, Johannes Heidecke, John Fishbein, John Hallman, Jonah Grant, Jonathan Chien, Jonathan Gordon, Jongsoo Park, Jordan Liss, Jos Kraaijeveld, Joseph Guay, Joseph Mo, Josh Lawson, Josh McGrath, Joshua Vendrow, Joy Jiao, Julian Lee, Julie Steele, Julie Wang, Junhua Mao, Kai Chen, Kai Hayashi, Kai Xiao, Kamyar Salahi, Kan Wu, Karan Sekhri, Karan Sharma, Karan Singhal, Karen Li, Kenny Nguyen, Keren Gu-Lemberg, Kevin King, Kevin Liu, Kevin Stone, Kevin Yu, Kristen Ying, Kristian Georgiev, Kristie Lim, Kushal Tirumala, Kyle Miller, Lama Ahmad, Larry Lv, Laura Clare, Laurance Fauconnet, Lauren Itow, Lauren Yang, Laurentia Romaniuk, Leah Anise, Lee Byron, Leher Pathak, Leon Maksin, Leyan Lo, Leyton Ho, Li Jing, Liang Wu, Liang Xiong, Lien Mamitsuka, Lin Yang, Lindsay McCallum, Lindsey Held, Liz Bourgeois, Logan Engstrom, Lorenz Kuhn, Louis Feuvrier, Lu Zhang, Lucas Switzer, Lukas Kondraciuk, Lukasz Kaiser, Manas Joglekar, Mandeep Singh, Mandip Shah, Manuka Stratta, Marcus Williams, Mark Chen, Mark Sun, Marselus Cayton, Martin Li, Marvin Zhang, Marwan Aljubeh, Matt Nichols, Matthew Haines, Max Schwarzer, Mayank Gupta, Meghan Shah, Melody Huang, Meng Dong, Mengqing Wang, Mia Glaese, Micah Carroll, Michael Lampe, Michael Malek, Michael Sharman, Michael Zhang, Michele Wang, Michelle Pokrass, Mihai Florian, Mikhail Pavlov, Miles Wang, Ming Chen, Mingxuan Wang, Minnia Feng, Mo Bavarian, Molly Lin, Moose Abdool, Mostafa Rohaninejad, Nacho Soto, Natalie Staudacher, Natan LaFontaine, Nathan Marwell, Nelson Liu, Nick Preston, Nick Turley, Nicklas Ansman, Nicole Blades, Nikil Pancha, Nikita Mikhaylin, Niko Felix, Nikunj Handa, Nishant Rai, Nitish Keskar, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Oona Gleeson, Pamela Mishkin, Patryk Lesiewicz, Paul Baltescu, Pavel Belov, Peter Zhokhov, Philip Pronin, Phillip Guo, Phoebe Thacker, Qi Liu, Qiming Yuan, Qinghua Liu, Rachel Dias, Rachel Puckett, Rahul Arora, Ravi Teja Mullapudi, Raz Gaon, Reah Miyara, Rennie Song, Rishabh Aggarwal, RJ Marsan, Robel Yemiru, Robert Xiong, Rohan Kshirsagar, Rohan Nuttall, Roman Tsiupa, Ronen Eldan, Rose Wang, Roshan James, Roy Ziv, Rui Shu, Ruslan Nigmatullin, Saachi Jain, Saam Talaie, Sam Altman, Sam Arnesen, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Sarah Yoo, Savannah Heon, Scott Ethersmith, Sean Grove, Sean Taylor, Sebastien Bubeck, Sever Banesiu, Shaokyi Amdo, Shengjia Zhao, Sherwin Wu, Shibani Santurkar, Shiyu Zhao, Shraman Ray Chaudhuri, Shreyas Krishnaswamy, Shuaiqi, Xia, Shuyang Cheng, Shyamal Anadkat, Simón Posada Fishman, Simon Tobin, Siyuan Fu, Somay Jain, Song Mei, Sonya Egoian, Spencer Kim, Spug Golden, SQ Mah, Steph Lin, Stephen Imm, Steve Sharpe, Steve Yadlowsky, Sulman Choudhry, Sungwon Eum, Suvansh Sanjeev, Tabarak Khan, Tal Stramer, Tao Wang, Tao Xin, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Degry, Thomas Shadwell, Tianfu Fu, Tianshi Gao, Timur Garipov, Tina Sriskandarajah, Toki Sherbakov, Tomer Kaftan, Tomo Hiratsuka, Tongzhou Wang, Tony Song, Tony Zhao, Troy Peterson, Val Kharitonov, Victoria Chernova, Vineet Kosaraju, Vishal Kuo, Vitchyr Pong, Vivek Verma, Vlad Petrov, Wanning Jiang, Weixing Zhang, Wenda Zhou, Wenlei Xie, Wenting Zhan, Wes McCabe, Will DePue, Will Ellsworth, Wulfie Bain, Wyatt Thompson, Xiangning Chen, Xiangyu Qi, Xin Xiang, Xinwei Shi, Yann Dubois, Yaodong Yu, Yara Khakbaz, Yifan Wu, Yilei Qian, Yin Tat Lee, Yinbo Chen, Yizhen Zhang, Yizhong Xiong, Yonglong Tian, Young Cha, Yu Bai, Yu Yang, Yuan Yuan, Yuanzhi Li, Yufeng Zhang, Yuguang Yang, Yujia Jin, Yun Jiang, Yunyun Wang, Yushi Wang, Yutian Liu, Zach Stubenvoll, Zehao Dou, Zheng Wu, and Zhigang Wang. Openai gpt-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.

Vali Tawosi, Salwa Alamir, Xiaomo Liu, and Manuela Veloso. Meta-rag on large codebases using code summarization, 2025. URL https://arxiv.org/abs/2508.02611.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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,

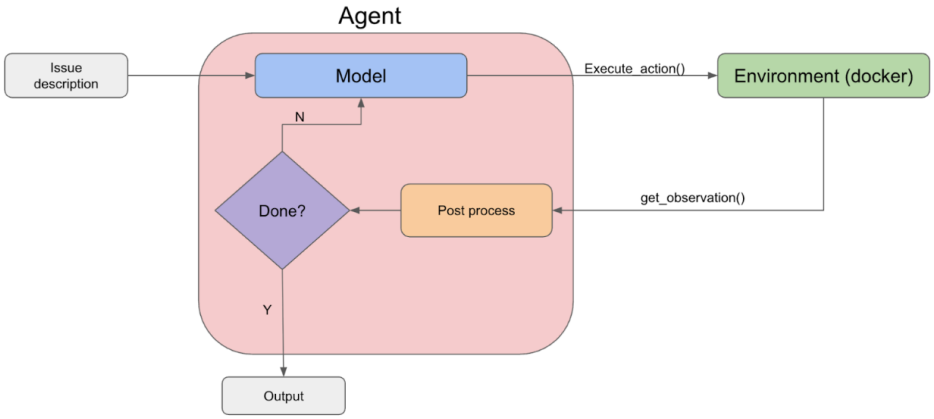

Figure 3: A block diagram of the mini-swe-agent framework.

Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025a. URL https://arxiv.org/abs/2407.16741.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. Coderag-bench: Can retrieval augment code generation?, 2025b. URL https://arxiv.org/abs/2406.14497.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

A Appendix

A.1 SWE-bench benchmark description

SWE-bench is a benchmark designed to evaluate large language models on complex real-world software engineering tasks drawn from GitHub (Jimenez et al., 2024). Given a repository and a GitHub issue, a model must produce a patch that resolves the problem described. Each instance in the dataset consists of a failing test (Fail-to-Pass) that is fixed by the corresponding pull request along with additional tests to ensure no unrelated behavior is broken (Pass-to-Pass). Each example in the benchmark contains a "gold" patch file, which has the code which are able to resolve the problem mentioned in the issue description and pass both suites of tests. The tasks come from actual GitHub issues and their associated fixes, making the evaluation grounded in how software evolves in practice. While many recent works focus on SWE-bench in the context of agentic / file-retrieval workflows (given its complexity), we set out to explore its suitability as a long-context benchmark, considering how it uses retrieved context from the entire repositories as the context.

The SWE-bench evaluation pipeline has two main stages: **generation** and **evaluation**. In the generation stage, the model is prompted with an issue description and relevant repository context, and it produces a candidate patch. In the evaluation stage, this patch is tested inside a controlled Docker environment: 1. the repository is cloned, 2. the model's patch is applied to the targeted file(s), and 3. the full test suite is run to check whether the issue is resolved without breaking other functionality.

A.2 mini-swe-agent Diagram

In Figure 3, we show a basic flowchart of the control flow of the agentic harness.

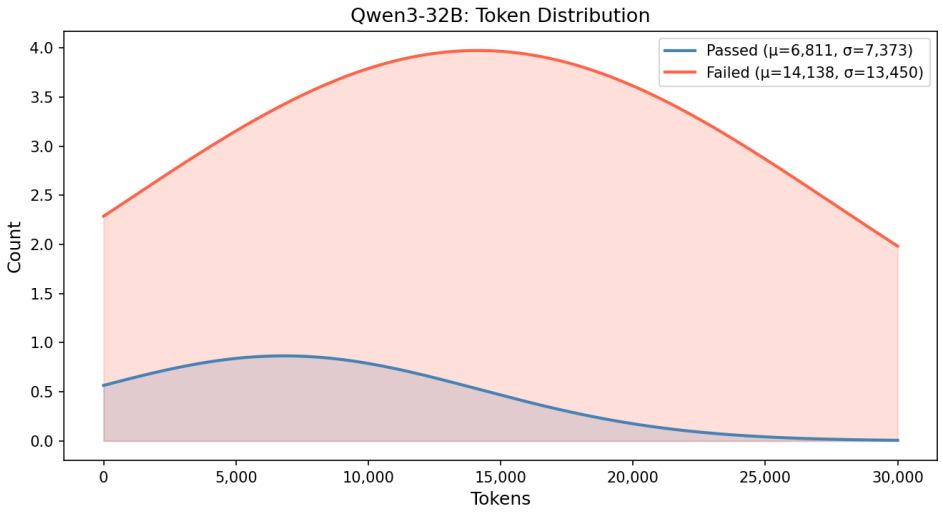

Figure 4: Token distribution of Qwen3-32B. The passed category includes 16 samples whereas the failed category includes 88 samples.

A.3 QWEN3-32B TOKEN DISTRIBUTION

In figure 4, we show the token distribution of the traces with Qwen3-32B on both the passed and failed categories. One difference that we noted compared to Deepseek-R1-0528 and GPT-5-nano is that we had many more samples which seemed to have failed so we increased the sample set from 100 to 150 and ended up with 104 valid runs. Ultimately, we still observe a similar trend that failed trajectories consume more tokens than successful ones and that the vast majority are under 20-30k tokens.