Sonar Foundation Agent Technical Report

Haifeng Ruan
SonarSource
haifeng.ruan@sonarsource.com

Yuntong Zhang
SonarSource
zhang.yuntong@sonarsource.com

Abstract

This report contains detailed description and analysis of Sonar Foundation Agent.

1 Introduction

Sonar Foundation Agent is a coding agent for general software issues, developed at Sonar by the former AutoCodeRover team. As of December 19th, 2025, Sonar Foundation Agent scores 79.4% on SWE-bench Verified and 52.62% on SWE-bench, while maintaining a low average cost of $1.98 and a high efficiency of 10.4 minutes per issue.

Quick clarification for Sonar customers The Sonar Foundation Agent is an internal project rather than a commercial product. The technology behind it, however, will show up in our real offering — the SonarQube Remediation Agent, which is currently in beta.

2 Design & Implementation

2.1 General Architecture

Sonar Foundation Agent is a tool-calling-style agent, implemented with the LlamaIndex [1] framework. Configured with a carefully-designed system prompt, Sonar Foundation Agent receives the description of the issue to solve and then iteratively invokes tools to investigate and resolve the issue. The final output is a patch to the code in the unified diff format.

2.2 Tools

Tool Choice In Sonar Foundation Agent, we employ two simple and effective tools: the bash and the text editor. Both tools have proved to work well for a variety of coding agents. More importantly, the two tools are deeply integrated into the Anthropic models, with dedicated tool names (`bash_20250124` and `text_editor_20250728`). Therefore, it is likely that the Anthropic models have been trained for both tools, yielding additional efficacy gains. Additionally, Sonar Foundation Agent also uses the AST (abstract syntax tree) Search tool of AutoCodeRover [4]. When the agent needs to look for the definition of symbols in the codebase, the AST Search tool is often a more efficient way than using `grep` from the bash tool, saving the number of reasoning steps and LLM costs. The agent is allowed at most 150 steps for a task, which was decided empirically to achieve a balance between efficacy and cost.

Bash To exploit the full power of the Anthropic models, our bash tool has an interface compatible with Anthropic's `bash_20250124`. The timeout of each bash command is set to 5 minutes, after which the command will be terminated and only the output collected within the timeout will be returned.

Text Editor Similar to the bash tool, our editor tool is also compatible with Anthropic's interface, `text_editor_20250728`. It can view the directory structure and files, create files, and edit files by

string replacement. After each successful edit, the updated portion of the file is returned for the agent to verify.

AST Search The AST Search tool is similar to that of AutoCodeRover, capable of searching for a class or function in a certain file or the whole codebase. An index of the classes and functions in the codebase is built at the beginning of the agent execution, by parsing the whole codebase with treesitter.

2.3 Prompts

Sonar Foundation Agent uses only two prompts: a system prompt defining the methodology for solving issues, and a user prompt defining the task and the output format.

System Prompt We design the system prompts with a few considerations:

1. Test-driven methodology. We stipulate the core principles of a test-driven methodology, including fully understanding the issue, writing a reproducer test, fixing the issue, and verifying the issue with the reproducer and regression tests[1].
2. High-level instructions. The methodology is outlined at a high level as core principles rather than step-by-step instructions, to allow thinking models the autonomy needed for solving the issues in a creative way. Additionally, the system prompt explicitly encourages a flexible approach, e.g., "adapt your approach to task complexity" and "whether through reproduction, code inspection, or test analysis, ensure you're fixing the right thing."
3. No overfitting to SWE-bench. For Sonar Foundation Agent to be useful in practice, we make sure to avoid any knowledge specific to the benchmark, and only include generally applicable software engineering principles in the prompt.

User Prompt The user prompt provides the agent with the issue description from SWE-bench verbatim. It also specifies the expected output format, that the agent is to save the patch to a particular location on the disk. In particular, the agent is encouraged to save tentative patches produced in the process of solving the issue. Otherwise, the agent would often run out of reasoning steps while trying to validate a tentative patch, thus forced to terminate without giving an answer.

3 Evaluation

3.1 Efficacy and Cost

We evaluate the efficacy of Sonar Foundation Agent on both SWE-bench Verified and SWE-bench Full. SWE-bench Verified [3] is a widely used benchmark for evaluating repository-level LLM agents for resolving issues in popular open-source software repositories. It contains 500 human-validated issues that contain well-specified issue descriptions. SWE-bench Full [3] is a superset of SWE-bench Verified, containing 2294 repository issues. Compared to SWE-bench Verified, SWE-bench Full has received less participation on the official SWE-bench leaderboard [2]. We evaluate Sonar Foundation Agent on both benchmarks to understand its performance both when the issues are well-specified and when the issues can be potentially under-specified.

Table 1 presents the efficacy of Sonar Foundation Agent. On SWE-bench Verified, Sonar Foundation Agent achieves a resolution rate of 79.2% with Claude Opus-4.5. Sonar Foundation Agent generally has higher efficacy on Claude models, compared to GPT-5 and Gemini 3 Pro. We attribute this partly to the design of Sonar Foundation Agent, as the bash tool and text editor tool are more closely related to Anthropic models. On the other hand, GPT-5 and Gemini 3 Pro incurs much lower costs with Sonar Foundation Agent, compared to Claude models. For example, Sonar Foundation Agent + GPT-5 resolved 70.8% of issues in SWE-bench Verified with an average cost of $0.45. The efficacy and cost present a trade-off when deploying Sonar Foundation Agent in different usage scenarios.

[1]When performing evaluation on SWE-bench, we strictly follow the rules and avoid using metadata about regression tests, i.e., `PASS_TO_PASS` and `FAIL_TO_PASS`. Instead, Sonar Foundation Agent can discover and run the relevant regression tests fully autonomously.

On SWE-bench Full consisting 2294 issues, Sonar Foundation Agent resolved 52.8% of the issues with Claude Opus-4.5, which is the new state-of-the-art on SWE-bench Full. While agents achieve approximately 80% efficacy on SWE-bench Verified, a significant performance gap remains on SWE-bench Full. This disparity highlights the need to enhance LLM agents' ability to automatically resolve ambiguity in under-specified natural-language issue descriptions, which more closely mirror the complexities of real-world production environments.

Table 1: Efficacy and cost of Sonar Foundation Agent on SWE-bench.

Tool	LLM	Resolved%	Average Cost ($)
SWE-bench Full			
Sonar Foundation Agent	Claude Opus 4.5	52.8%	1.98
SWE-bench Verified			
Sonar Foundation Agent	GPT-5	70.8%	0.45
	Gemini 3 Pro	72.4%	0.59
	Claude Sonnet 4.5	74.8%	1.25
	Claude Opus 4.5	79.2%	1.90

3.2 Patch Analysis

Patch Size We further analyze the size of patches generated by Sonar Foundation Agent on SWE-bench Verified, with different LLMs as the backend. Figure 1, 2, 3 present the patch churn distribution (in log scale) from GPT-5, Sonnet-4.5, and Opus-4.5, respectively. Each figure plots the distribution of agent-generated patches based on their sizes, where the patch size is defined by the number of lines added and removed. Additionally, each bar in the figure shows both the correct and incorrect patches, where correctness here is defined by whether the patch resolves the issue. Across all three LLM backends, the majority of the agent-generated patches have less than 30 changed lines. The ratio of correct patches is also higher when the patch size is smaller. These observations suggest that the smaller patches generated by the agent have a high probability of resolving the issue compared to larger patches. While all three LLMs exhibit a similar distribution of patch size, GPT-5 patches are more evenly distributed across the 1–20 range. In contrast, both Sonnet 4.5 and Opus 4.5 show a higher concentration of patches in the 2–4 size range.

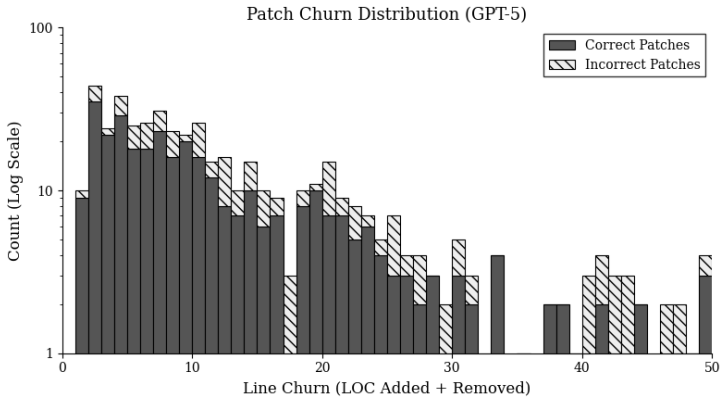

Figure 1: Patch Churn Distribution with Sonar Foundation Agent + GPT-5.

Exact Match Analysis To understand the effect of potential LLM memorization, we examine the exact match rate of agent-generated patches. An agent-generated patch is considered as an exact match, if it makes the same code modification (i.e., excluding comments etc.) as the ground truth developer's patch. Table 2 shows the exact match rate of patches generated by Sonar Foundation Agent with different LLM backends. Even with the same agent scaffold, LLMs show a large variation in the exact match rate, with GPT-5 at 11.49% and Opus-4.5 at 23%. However, we note that the

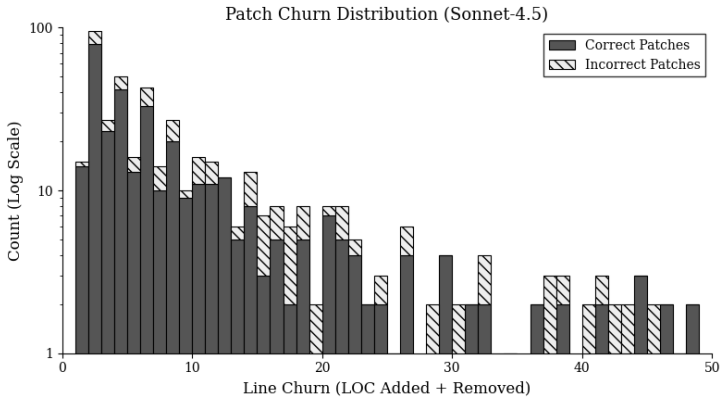

Figure 2: Patch Churn Distribution with Sonar Foundation Agent + Sonnet-4.5.

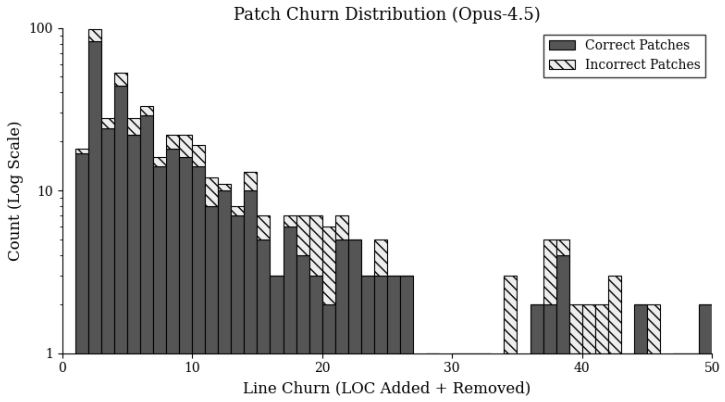

Figure 3: Patch Churn Distribution with Sonar Foundation Agent + Opus-4.5.

exact-match patches are usually the smaller patches. Table 2 also presents the average size of the exact-match patches, and the average size of all patches generated with different LLMs as backend. The exact-match patches on average modified 4.02 to 5.22 lines, while patches in general modified 12.44 to 15.32 lines.

4 Lesson learned: Tailoring agent autonomy

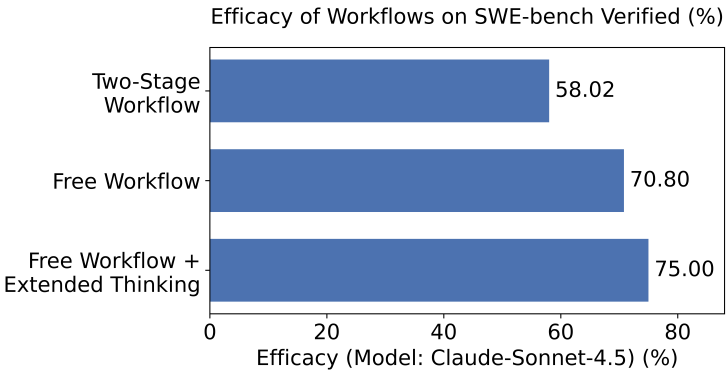

Figure 4: Efficacy increases with the level of agent autonomy.

Table 2: Percentage of agent-generated patches that are exact match to developer's patch.

LLM	Exact Match%	Avg. Patch Size (Exact Match)	Avg. Patch Size (All Patches)
GPT-5	11.49%	4.02	15.32
Claude Sonnet 4.5	18.20%	5.22	14.78
Claude Opus 4.5	23.00%	4.80	12.44

In trying to improve the efficacy of Sonar Foundation Agent, our team has conducted extensive research and experiments. During this process, it became clear to us that the key to a great agent is to match the level of autonomy with the capability of the underlying model. As shown in Figure 4, with the current powerful LLMs, the efficacy of our agent on SWE-bench Verified increases when given more autonomy.

4.1 Constrained workflow: Early-days AutoCodeRover

Back in April 2024, our team developed AutoCodeRover, which was one of the earliest coding agents. It has a clearly-defined 2-stage workflow: context retrieval, followed by patch generation. Either stage is handled by a separate agent. In the context retrieval stage, AutoCodeRover would repeatedly invoke an AST search tool to find the buggy location and accumulate relevant context, and most of the autonomy of AutoCodeRover lies in what AST searches to perform and when to stop. In the patch generation stage, AutoCodeRover would simply write a patch with the accumulated context. There is no autonomy in deciding the workflow.

The limited autonomy to AutoCodeRover was a conscious decision. At that time, the capability of LLMs to grasp a long context was limited. When we instructed a single agent to first retrieve context and then write a patch, it would have lost sight of some context collected early on when writing the patch. Moreover, oftentimes, it would not write a patch altogether. Therefore, we broke the workflow into two distinct stages: the context is first collected and summarized by a first agent, and a patch is written by a second agent. We found that the separation improved both context utilization and instruction following, boosting AutoCodeRover's efficacy under limited LLM capability.

4.2 More autonomy in workflow and tools: Sonar Foundation Agent

This time around, while developing Sonar Foundation Agent, we re-examined AutoCodeRover's two-stage workflow and its basis. We realized that the capability of LLMs have evolved a lot over the past year and a half, and that the agent might now benefit from a more free workflow. Therefore, we switched to a single-agent workflow. The two-stage workflow was not totally discarded. Instead, we prompted the single agent to work in several stages, including the two stages and more patch testing and validation. We were glad to find that the latest models, including GPT-5 and Claude Sonnet 4.5, are able to deal with a longer context window and follow instructions significantly better. Using the same LLM, the change in workflow resulted in an efficacy boost from about 58% ("Two-Stage Workflow" in Figure 4) to 70% ("Free Workflow" in Figure 4).

4.3 More autonomy in prompts: Leveraging thinking models

Finally, we sought to unlock the power of thinking models. In an initial attempt, we simply turned on the extended thinking of Claude Sonnet 4.5. However, the efficacy remained at about 70%. We realized that the prompt was so detailed that even with extended thinking, the agent would do largely the same things and achieve similar results. Our realization was corroborated by Claude's official prompting guide, which says thinking models can benefit from more concise and less prescriptive prompts. In light of this, we distilled the essence of our prompt, highlighting a test-driven approach to issue resolving, while removing the overly prescriptive instructions. This improvement in prompts gave us a final boost of efficacy to 75% ("Free Workflow+Extended Thinking" in the chart above).

The journey from AutoCodeRover to the Sonar Foundation Agent offers a critical insight for the future of agentic coding: as underlying models grow more powerful, we must grant them more autonomy. Our research clearly shows that moving from a constrained, two-stage process to a "Free Workflow" and refining prompts to be less prescriptive unlocked the agent's full potential, boosting

efficacy from 58% to 75%. This principle of matching agent autonomy to model capability will be foundational as we continue to push the boundaries of AI-driven software development.

References

[1] Llamaindex - redefine document workflows with ai agents. `https://www.llamaindex.ai/`. Accessed: 2025-12-11.

[2] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Gaunt, Lakshman Niranjan, Ziyang Zhong, Karthik Narasimhan, and Anirudh Narasimhan. SWE-bench official leaderboards. `https://www.swebench.com/`, 2024. Accessed: 2025-12-15.

[3] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[4] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In Maria Christakis and Michael Pradel, editors, *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, pages 1592–1604. ACM, 2024.