Lingxi Technical Report

Implementation on SWE-bench – Version 1.0 – 7 May 2025

Abstract

Lingxi is an open-source, multi-agent framework designed to automate a broad range of software-engineering tasks—from code review and documentation generation to automated program repair. This report presents our use of Lingxi for automated bug fixing on the SWE-bench-Lite benchmark. By decomposing the repair workflow into specialised agents, Lingxi achieves a **42.67 % issue-resolution rate (128 / 300) on the SWE-bench-Lite test set**, ranking **#3 among open-source models and #7 overall**. We detail the system design, tool design, experimental results, and planned improvements in below.

1 Introduction

Lingxi is a **general-purpose, multi-agent framework for software engineering tasks** build on top of LangGraph, all module of the Lingxi are expandable. It can be configured for diverse tasks such as vulnerability detection, vulnerability exploitbility analysis ,test generation, or automatic program repair. In this report we describe our implementation that targets automated bug fixing on SWE-bench-Lite.

Single-agent pipelines often fail because the chat history grows uncontrollably and distracts the model during delicate code-editing steps. Lingxi solves this problem through a **task-scope multi-agent architecture**. Each agent handles a well-defined sub-task, while a lightweight coordinator coordinates their interaction.

2 System Overview

2.1 Motivation for a Multi-Agent Approach

Single-agent pipelines that keep the entire dialogue in one context inevitably suffer from **context dilution**: by the time the model reaches the code-editing step, earlier discussion tokens dominate the prompt, making it harder for the LLM to focus on the actual diff. Lingxi avoids this by splitting the workflow into compact, purpose-built agents that each receive only the information they need.

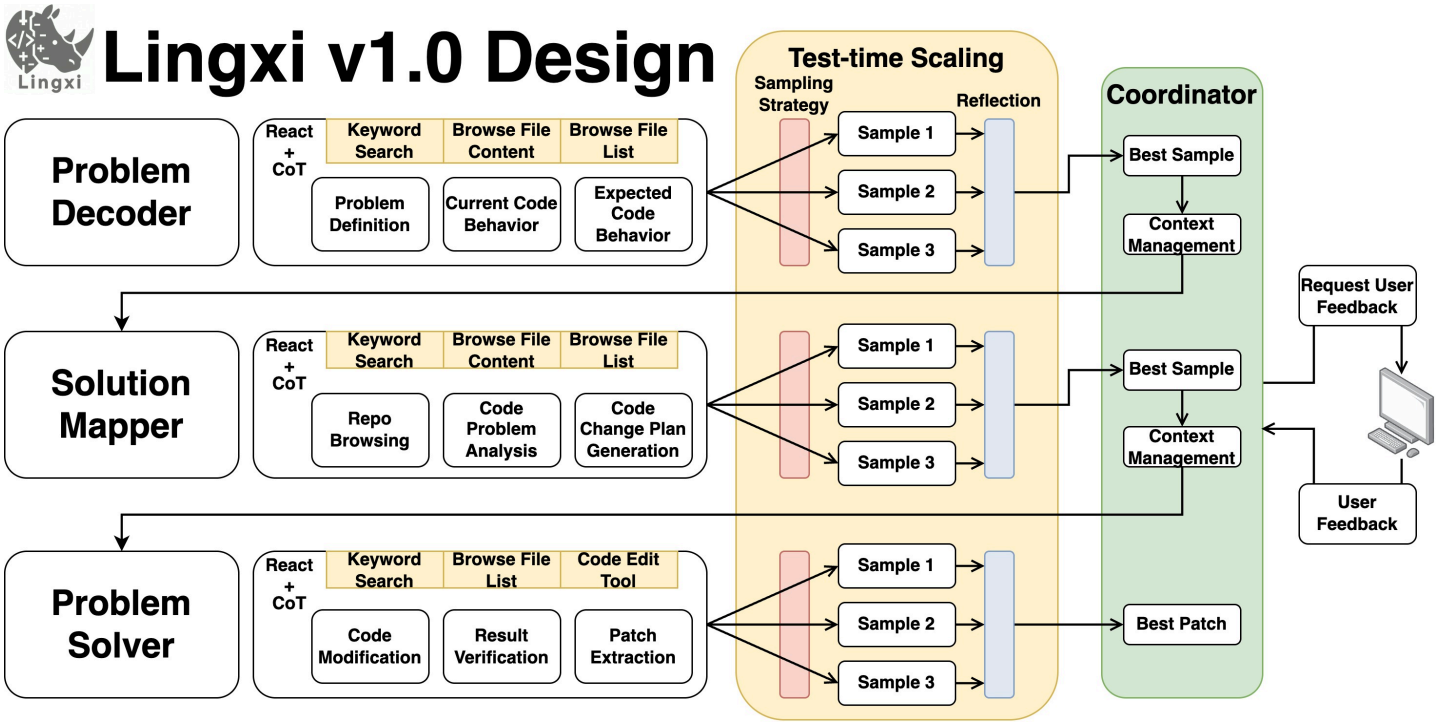
2.2 Design Challenges

A recent Berkeley study reports that naive multi-agent orchestration frequently underperforms single-agent baselines because of

1. **Vague task specifications**,
2. **Unclear responsibility boundaries**, and
3. **Chaotic memory or state management**.

We observed the same pitfalls while prototyping Lingxi and introduced targeted mitigations described below.

2.3 Lingxi Architecture



Agent	Core Question Answered	Responsibility
Problem Decoder	*What* is wrong and *where* is it?	Locate the root cause, report the failing scenario, and pinpoint files / functions.
Solution Mapper	*How* do we fix it and *why* will that work?	Produce an ordered, line-granular patch plan.
Problem Solver	*Do* the edit.	Apply the patch, run the full test suite, and capture logs.
Critic/Reflection	*Did* it work?	Validate and compare the agent's attempts, and request another iteration if needed.
coordinator	—	Govern agent lifecycles, enforce tool contracts, shrink context, and mediate messages.

Unlike frameworks that imitate human roles (e.g., *developer*, *manager*), Lingxi's roles are derived from **task decomposition** rather than social division of labour. Because every agent is backed by the *same* LLM, we gain little from "specialisation"; the win comes from narrowing each agent's task domain so the prompt is shorter, task is easier and the evaluation rubric is crisp.

2.4 Optimisations That Matter

1. **Crystal-clear contracts** – each agent gets a one-page interface spec listing mandatory inputs, expected outputs, and the *only* tools it may call.
2. **coordinator-led memory hygiene** – after every step, the coordinator trims conversation history to the triad *(action, concise observation, reflection)*, discarding verbose function-call dumps. This reduced average prompt length by ≈ 38 % without loss of signal.
3. **Explicit awareness of other agents** – every prompt reminds the agent of the team composition and urges it to stay within scope.

2.5 Reflection & Debate (Road-map)

Our current reflection mechanism chooses the best of three self-generated solutions. Inspired by Google's *Co-Scientist*, we plan to evolve this into a lightweight debate where agents critique and iteratively refine each other's reasoning—leveraging emergent group intelligence instead of mere task parallelism.

2.6 Beyond SWE-bench

Lingxi's agent prompts, tool abstractions, and coordinator skeleton are **task-agnostic**. Lingxi is still under develop, in our vision, porting the framework to other SE tasks like code review, documentation generation requires minimal effort, like swapping task-specific prompts and—if necessary—adding domain-specific tools. We will release Lingxi's implementation on other SE tasks soon.

3 Tooling Design

Lingxi follows a **"minimal tool set, maximal information"** philosophy. We aim to provide accurate, sufficient information to LLM for each tool calling result, so that reduce the burden/steps for LLM. Below are the tools we design for SWE-Bench, all tools are exposed through structured function-calling so the LLM can chain them autonomously.

Tool	Purpose	Key Features
`view_directory`	Explore repository tree	Adaptive depth: prints deeper until a **file-count cap** is hit.
`search_files_by_keywords`	Grep-like semantic search	Multi-keyword queries via *ripgrep*; returns line numbers.
`view_file_content`	Inspect file text	Show file content, auto-truncates long files and appends a file structure.
`view_file_structure`	Summarise oversized files	**Invisible** to LLM; automatically injects when needed.
`str_replace_editor`	Apply code edits	Inspired by Anthropic *computer*, Aider, and OpenHand; `undo` removed for simplicity.

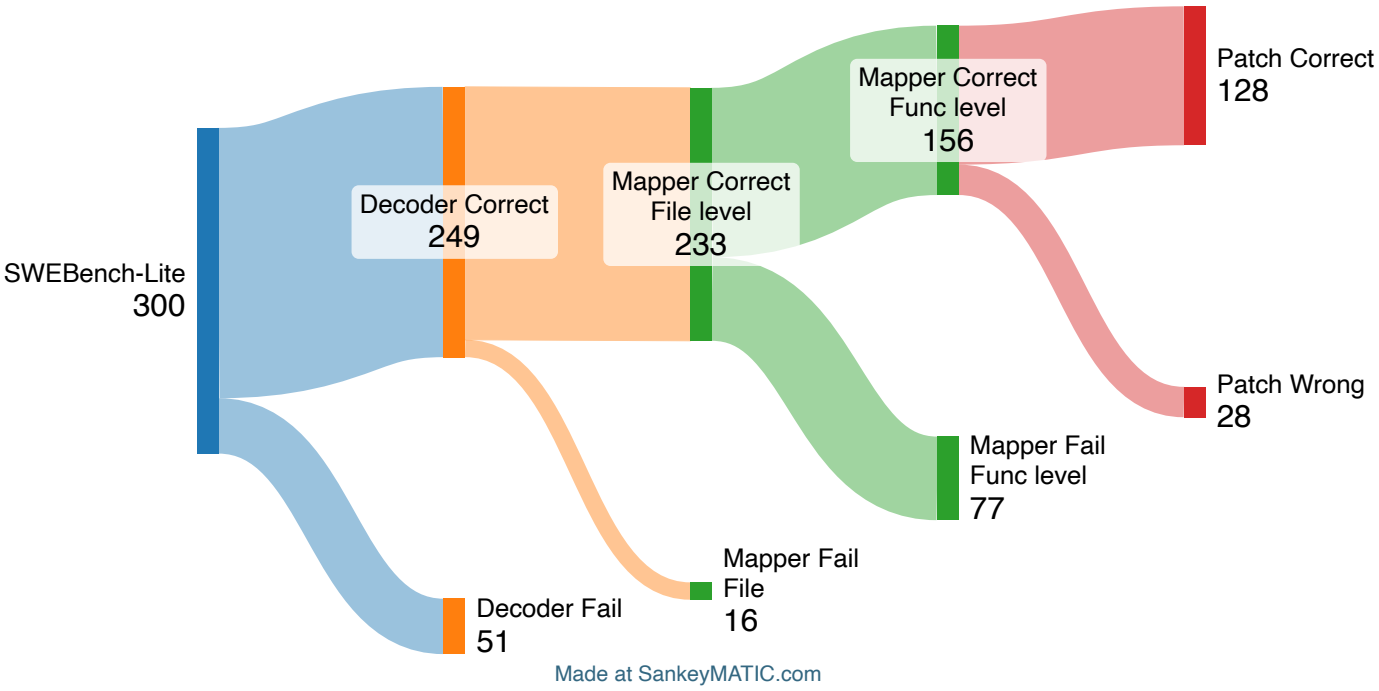
4 Results

4.1 End-to-End Performance

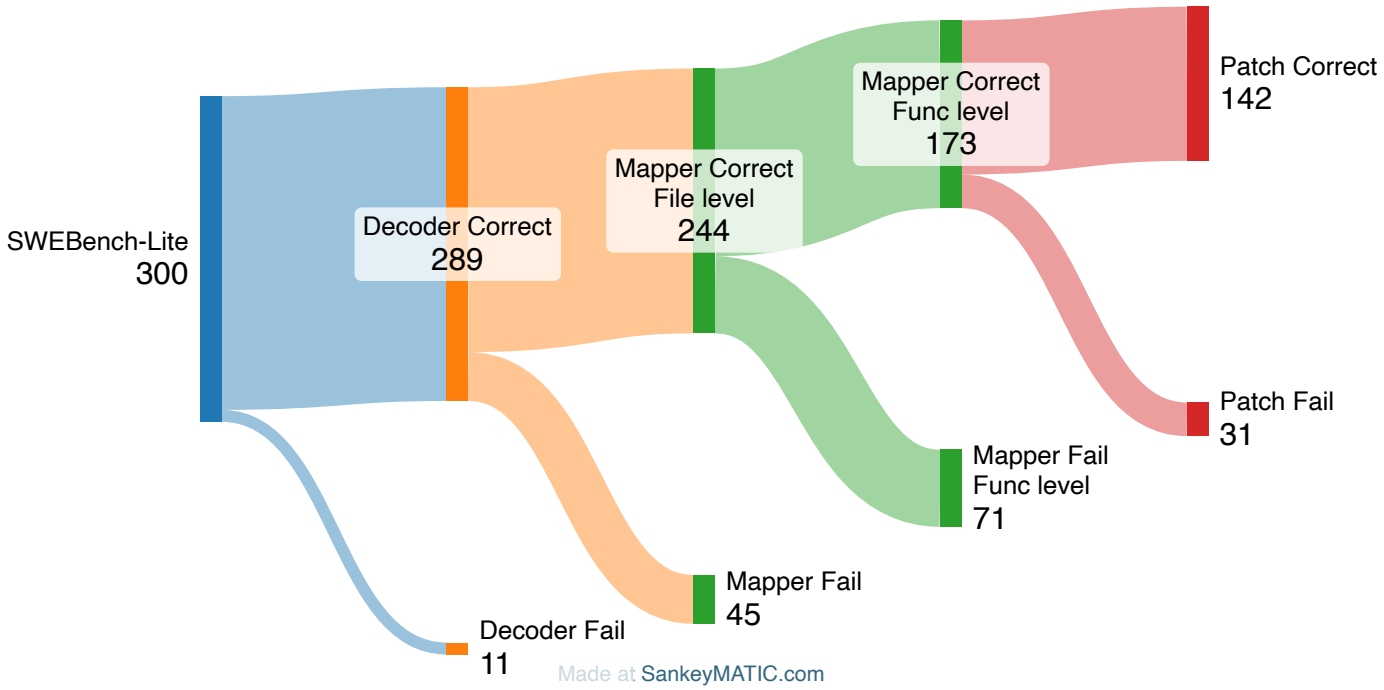
Metric	Score
Resolved issues	**132 / 300** (44.15 %)
SWE-bench-Lite OSS leaderboard	**3rd**
SWE-bench-Lite overall leaderboard	**7th**

4.2 Stage-wise Success Rates

- Critic selected:



- Any generated:



6 Planned Improvements (Version 2 Roadmap)

1. **Adaptive sampling for expensive evaluations**
    - Curate a balanced 60-instance subset that (a) preserves leaderboard order, (b) contains difficult and easy cases, and (c) includes all instances solved by rivals but missed by us.
2. **Reflection-driven debate**
    - Replace three-way self-reflection with a *debate* mechanism inspired by Google *Co-Scientist*, encouraging agents to critique and refine each other's reasoning.

7 Discussion

7.1 When Does Multi-Agent Help?

Our results confirm Berkeley's observation that naïve multi-agent systems often trail single-agent baselines. The key to outperforming them is **precise task scoping plus memory hygiene**. Removing large function-call dumps was especially impactful, reducing confusion and token cost.

8 Conclusion

Lingxi demonstrates that a carefully engineered multi-agent pipeline can surpass state-of-the-art open-source systems on SWE-bench-Lite. By design, Lingxi's architecture is **task-agnostic**; the same principles can be re-targeted to code review, vulnerability repair or other SE tasks .

References

1. **SWE-bench:** "Swe-bench: Benchmarking LLMs on Software Bug Fixing," *arXiv*, 2024.
2. **Google Co-Scientist:** Patel et al., "Improving reasoning via debate," *arXiv*, 2023.
3. **Anthropic Tool Use:** Chen et al., "Toolformer: Language models that can use tools," *arXiv*, 2023.