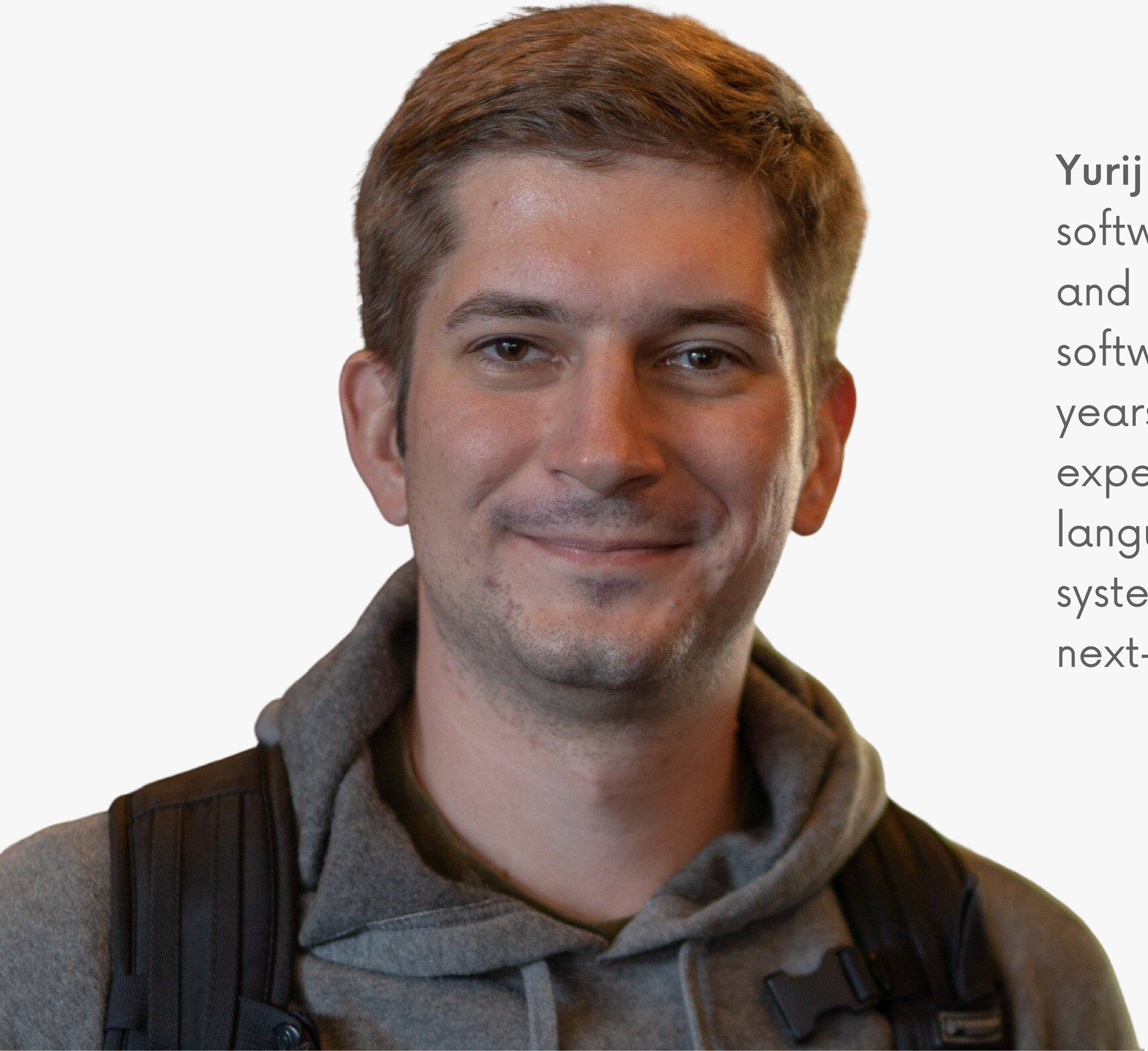
PRACTICAL APPLICATIONS OF STATE-OF-THE-ART LARGE LANGUAGE MODELS TO SOLVE REAL-WORLD SOFTWARE ENGINEERING PROBLEMS AUTONOMOUSLY

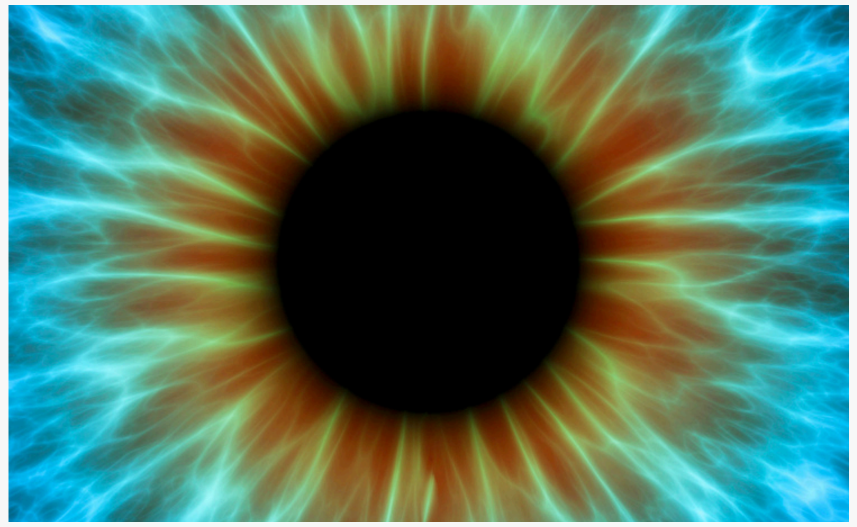
YURIJ MIKHALEVICH
QA WOLF
DUBAI, UNITED ARAB EMIRATES
EMAIL: YURIJ@MIKHALEVI.CH

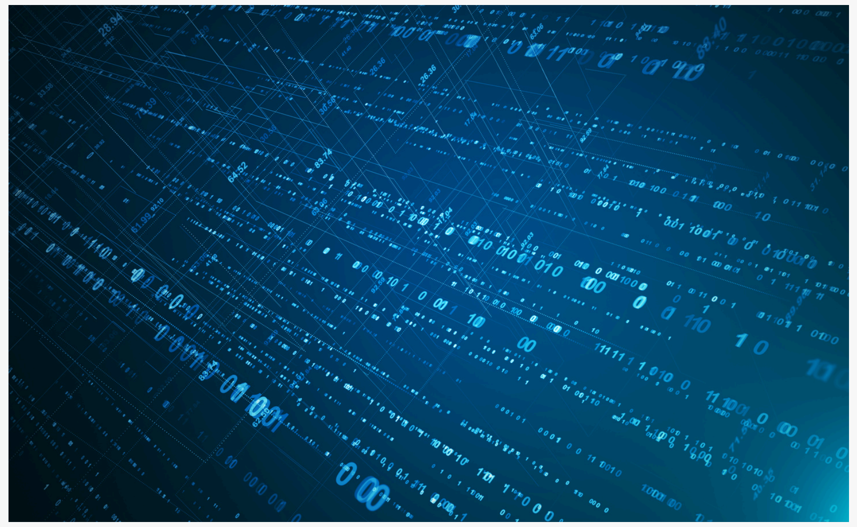


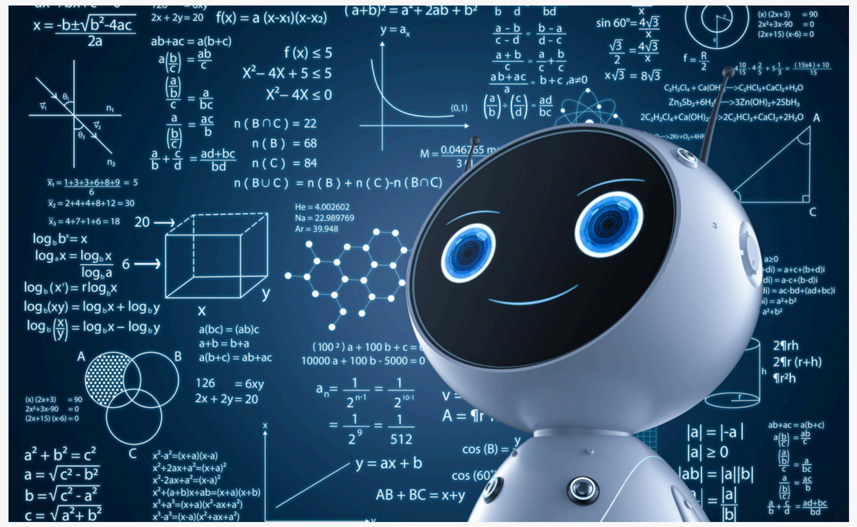
Yurij Mikhalevich, MSc Computer Science, is a software engineer, machine learning engineer, and researcher with over twelve years of industrial software engineering experience and over ten years of industrial machine learning engineering experience focusing on computer vision, natural language processing, and recommendation systems. Presently, he is building QA Wolf AI – the next-level AI expert at creating E2E tests.

RESEARCH INTERESTS

On the right are Yurij's current primary research interests.







Computer Vision

Both image and video processing, with the current focus on diffusion models and vision transformers.

Natural Language Processing

With a focus on recommendation systems and generative language models.

Reinforcement Learning

Systems that learn from the environment are fascinating.

INTRODUCTION

In this work, we explore the practical application of cutting-edge LLMs as autonomous software engineers on real-world tasks. We design an experiment in which an AI-driven coding agent is given only the natural-language description of a software issue (as one would find in a bug tracker or feature request) and is tasked with resolving the issue by modifying the codebase without human assistance. We evaluate the following state-of-the-art LLMs in this autonomous setting: **Claude Sonnet 3.7**, **DeepSeek-V3**, **DeepSeek-R1**, and **o3-mini-high**. We have used the **Aider** agent to solve problems – one of the best open-source AI software engineering agents. Additionally, we have evaluated the **Claude Code** agent as one of the best closed-source AI software engineering agents. We examine not only whether the LLM-powered agent can produce a working solution but also the quality of the solution (linting, code style, user experience) and the cost of the API calls.

RELATED WORKS: LLMS FOR CODE GENERATION AND ASSISTANCE

The use of large neural models for code generation has rapidly progressed in recent years. OpenAI's Codex model, which powers GitHub Copilot, was among the first to demonstrate that an LLM trained on vast amounts of code can produce syntactically correct and often functionally correct code for given descriptions. Subsequent models have pushed these capabilities further: DeepMind's AlphaCode achieved performance on par with average human competitors in programming contests, signaling the potential of LLMs to handle complex algorithmic problems. Recent developments in the field have demonstrated significant progress in computational capabilities. Specifically, models such as OpenAI o3-mini-high, DeepSeek-R1, and Claude Sonnet 3.7 have established new performance standards. These advancements indicate the continued rapid evolution of LLM capabilities, with potential implications for fully autonomous software engineering agents.

METHOD: INTRO

Our research methodology is designed to evaluate each LLM’s ability to autonomously resolve real software issues under controlled conditions. We selected the open-source project Aibyss, a web-based AI competition game, as our testbed. Aibyss is a TypeScript project (Nuxt/Vue frontend with a Node.js backend using Prisma ORM) where users write AI bots to compete in a game. We chose Aibyss because it is a non-trivial codebase with realistic features and bugs, yet manageable in size. From Aibyss’s issue tracker, we picked ten issues that were open and well-described. These issues covered a range of feature requests and bug fixes and were labeled by us based on the perceived difficulty as “easy,” “medium,” or “harder”.

METHOD: TASK SELECTION

1. easy - “feat: draggable splitter between the code and the game screen should remember its position between the page reloads”
2. easy - “feat(rating): highlight top results in k/d, kills, deaths, and food eaten columns in the rating table”
3. easy - “chore(World): double the frequency of food spawns”
4. medium - “feat: allow turning off the bots of some users by setting the “inactive” field in the database on the user object to ‘true”’
5. medium - “feat: ensure that the game screen occupies all available free space to the right of the code editor”
6. medium - “feat(rating): add a new column to the rating table displaying the number of times the user submitted the code”
7. medium - “feat(sandbox): add an option to turn off sprites and replace them with circles to make debugging easier”
8. harder - “bug: fix the issue causing the bot code to submit when the user opens “API reference”
9. harder - “feat: add code versions and an option to revert to a previous version”
10. harder - “feat: surface bot execution errors to the user”

The actual GitHub issues with their descriptions can be found on the Aibyss project GitHub issues page: github.com/move-fast-and-break-things/aibyss/issues?q=is%3Aissue%20label%3Aai-agents-evaluation-2025-03

METHOD: AGENTS AND LLM VARIANTS

- Aider 0.75.2 + o3-mini-high 2025-01-31
- Aider 0.75.2 + DeepSeek-V3
- Aider 0.75.2 + DeepSeek-R1
- Aider 0.75.2 + Claude Sonnet 3.7 20250219
- Aider 0.78.0 + Claude Sonnet 3.7 20250219 with 32k thinking tokens – in this variant, we enabled the “thinking mode” in Aider (using v0.78.0 with thinking support for Claude 3.7)
- Claude Code 0.2.35 – Anthropic’s Claude Code is a proprietary agent with a CLI interface very similar to Aider’s that uses the Claude Sonnet 3.7 model under the hood; this can be seen as a closed-source counterpart to Aider, specifically tuned for Claude

METHOD: AUTONOMY AND STOPPING CRITERIA

We configured the agents to operate fully autonomously.
Aider was run with the --yes-always flag, meaning it would automatically apply its proposed actions. In the case of Claude Code, we approved all its prompts manually.
Each agent was allowed to iterate until it produced no further actions.
One exception to full autonomy was with the o3-mini-high model in Aider: often, this model did not automatically load the files it needed, and would ask the user to add certain files to its context. Whenever Aider+o3-mini-high requested a file, we manually added exactly that file (and no additional help), then let it continue. No other agent required such interventions.

METHOD: EVALUATION CRITERIA

- **Works (Yes/No)**: Did the changes address the issue from the end-user's perspective? For a feature request, this meant the new functionality worked as intended. For a bug, the erroneous behavior was fixed.
- **Linting Check Pass**: We ran the project's linting scripts. If the agent's final code did not pass them, we marked that as a quality issue.
- **User Experience (UX)**: We manually inspected if the solution introduced any noticeable UX problems (e.g., a feature works but has a confusing UI or performance lag).
- **Code Quality:** We reviewed the diffs to assess if the solution was implemented in a reasonable and maintainable way. Inefficiencies, unmaintainable code, and obvious bugs in the implementation were noted.

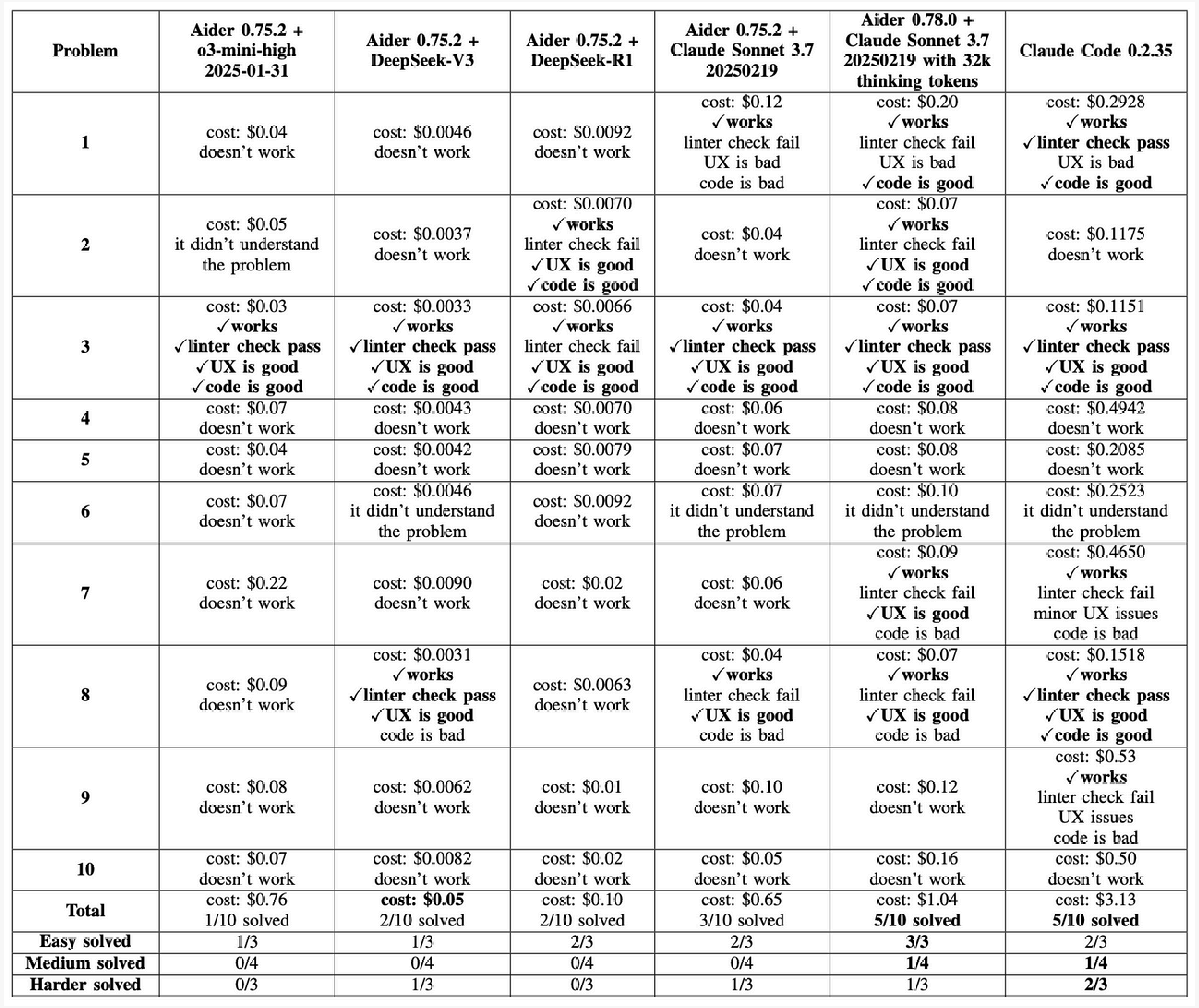
We selected these criteria because they mirror how work performed by a human software engineer is usually evaluated. These qualitative judgments were used to label each successful solution with additional notes (e.g., "works, but suboptimal code" or "works, except fails linting"). Finally, we measured the cost of each solution in USD.

IMPLEMENTATION

All agent runs were conducted in a consistent environment. We created a fresh Docker container for each run, which checked out the Aibyss repository at commit b4e58b2 (to ensure all models started from identical code) and installed the necessary tools (Node.js, Aider, Claude Code, etc.). The agent was then launched inside the container and given the issue text to solve. The prompt given to each agent was uniform: "Please solve the following issue. Title: <issue title> Description: <issue body>". We ensured the project's dependencies and database (SQLite for this test) were properly set up in each container so that the agent could run the app or tests if it chose to. The Aibyss codebase was about 3.5k lines of TypeScript/JavaScript. Each agent configuration was run on each of the 10 issues, yielding 60 trials in total.

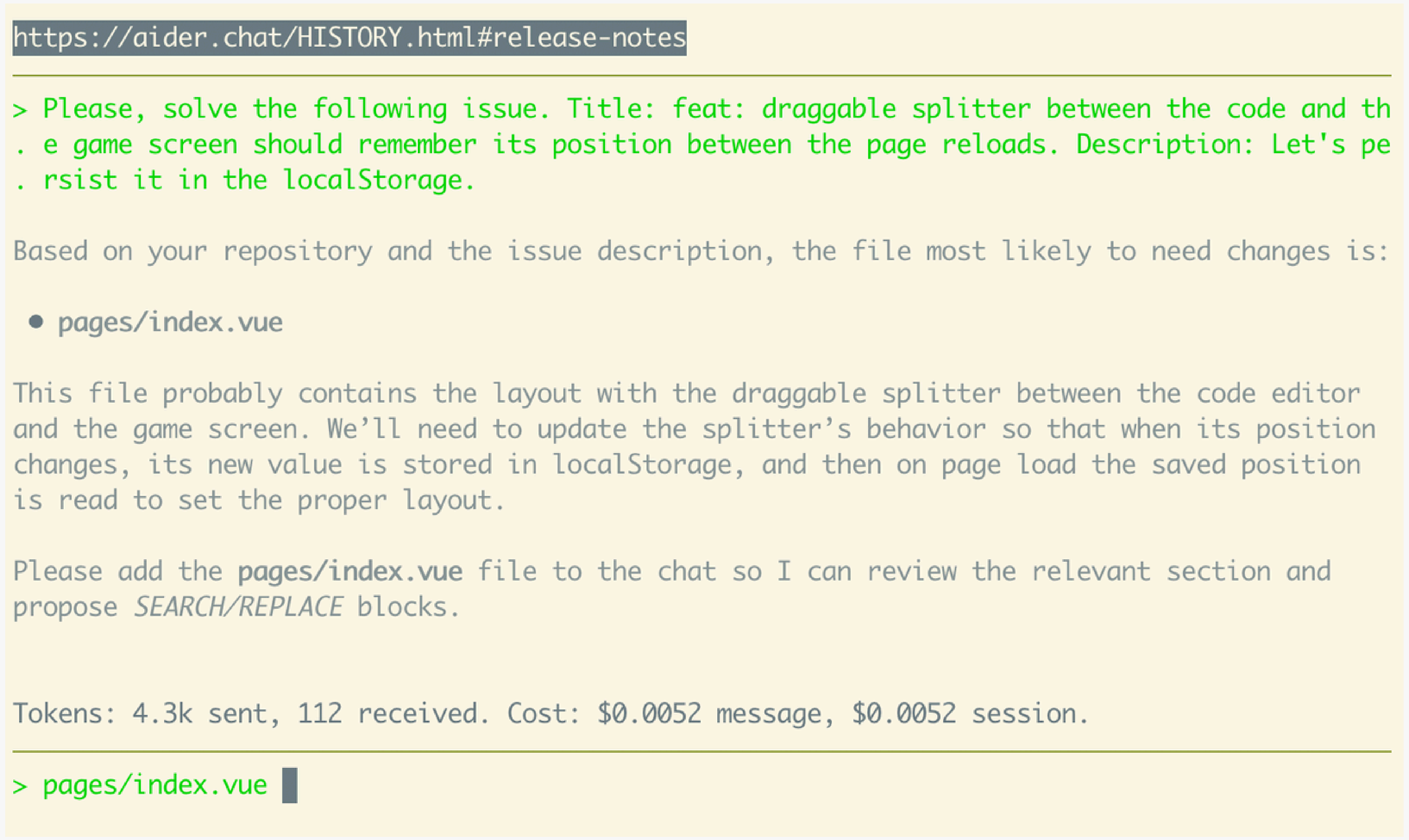
After an agent completed, we committed its changes to a new branch and opened a pull request on GitHub. This allowed us to use continuous integration (CI) results as an additional datapoint. We then manually reviewed and tested the branch as described in the evaluation criteria. All of the PRs created as part of this research can be found on GitHub: github.com/move-fast-and-break-things/aibyss/pulls?q=is%3Apr+label%3Aai-agents-evaluation-2025-03+

RESULTS



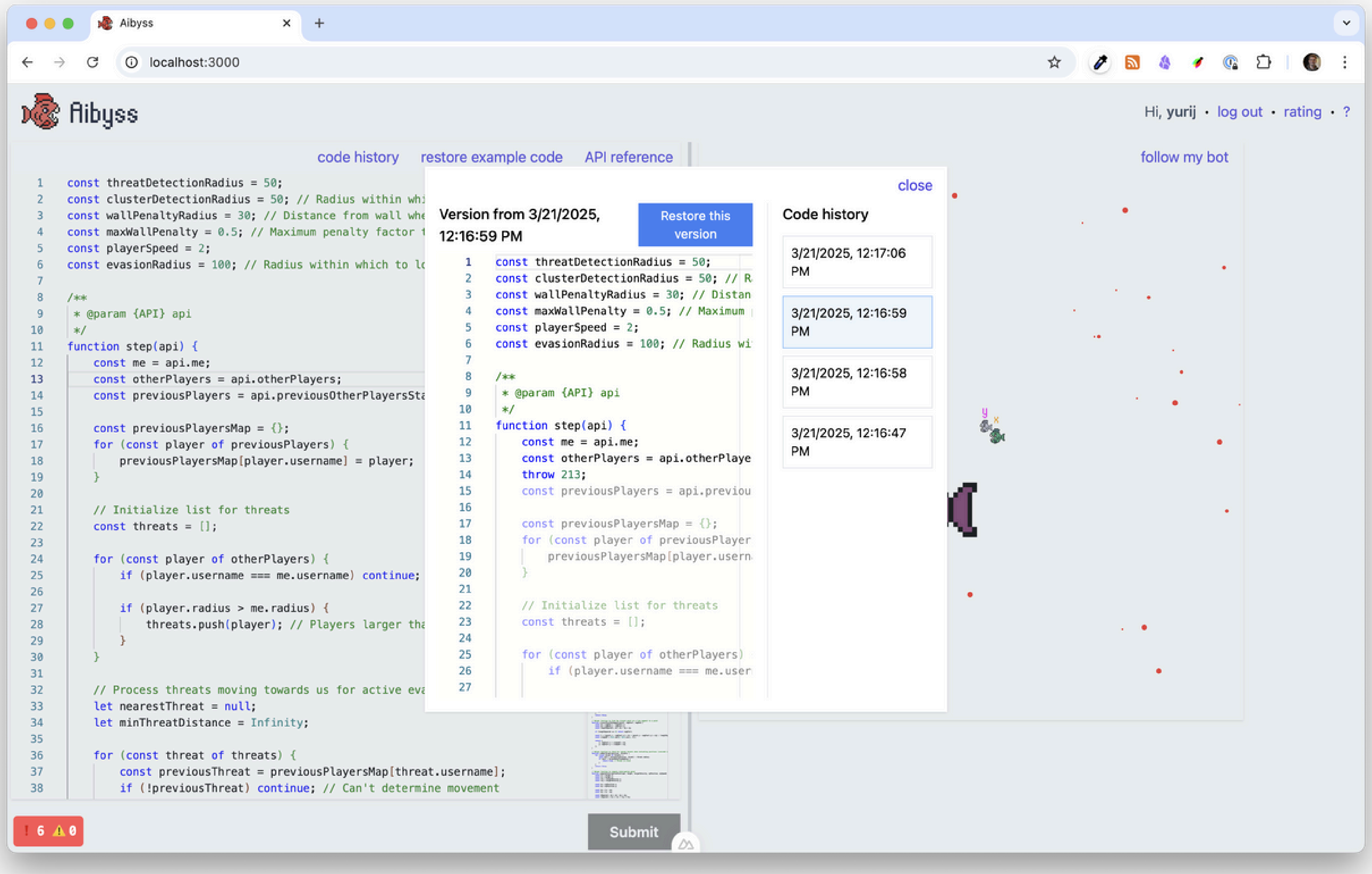
Problem	**Aider 0.75.2 + o3-mini-high 2025-01-31**	**Aider 0.75.2 + DeepSeek-V3**	**Aider 0.75.2 + DeepSeek-R1**	**Aider 0.75.2 + Claude Sonnet 3.7 20250219**	**Aider 0.78.0 + Claude Sonnet 3.7 20250219 with 32k thinking tokens**	**Claude Code 0.2.35**
1	cost: $0.04 doesn't work	cost: $0.0046 doesn't work	cost: $0.0092 doesn't work	cost: $0.12 **✓works** linter check fail UX is bad code is bad	cost: $0.20 **✓works** linter check fail UX is bad **✓code is good**	cost: $0.2928 **✓works** **✓linter check pass** UX is bad **✓code is good**
2	cost: $0.05 it didn't understand the problem	cost: $0.0037 doesn't work	cost: $0.0070 **✓works** linter check fail **✓UX is good** **✓code is good**	cost: $0.04 doesn't work	cost: $0.07 **✓works** linter check fail **✓UX is good** **✓code is good**	cost: $0.1175 doesn't work
3	cost: $0.03 **✓works** **✓linter check pass** **✓UX is good** **✓code is good**	cost: $0.0033 **✓works** **✓linter check pass** **✓UX is good** **✓code is good**	cost: $0.0066 **✓works** linter check fail **✓UX is good** **✓code is good**	cost: $0.04 **✓works** **✓linter check pass** **✓UX is good** **✓code is good**	cost: $0.07 **✓works** **✓linter check pass** **✓UX is good** **✓code is good**	cost: $0.1151 **✓works** **✓linter check pass** **✓UX is good** **✓code is good**
4	cost: $0.07 doesn't work	cost: $0.0043 doesn't work	cost: $0.0070 doesn't work	cost: $0.06 doesn't work	cost: $0.08 doesn't work	cost: $0.4942 doesn't work
5	cost: $0.04 doesn't work	cost: $0.0042 doesn't work	cost: $0.0079 doesn't work	cost: $0.07 doesn't work	cost: $0.08 doesn't work	cost: $0.2085 doesn't work
6	cost: $0.07 doesn't work	cost: $0.0046 it didn't understand the problem	cost: $0.0092 doesn't work	cost: $0.07 it didn't understand the problem	cost: $0.10 it didn't understand the problem	cost: $0.2523 it didn't understand the problem
7	cost: $0.22 doesn't work	cost: $0.0090 doesn't work	cost: $0.02 doesn't work	cost: $0.06 doesn't work	cost: $0.09 **✓works** linter check fail **✓UX is good** code is bad	cost: $0.4650 **✓works** linter check fail minor UX issues code is bad
8	cost: $0.09 doesn't work	cost: $0.0031 **✓works** **✓linter check pass** **✓UX is good** code is bad	cost: $0.0063 doesn't work	cost: $0.04 **✓works** linter check fail **✓UX is good** code is bad	cost: $0.07 **✓works** linter check fail **✓UX is good** code is bad	cost: $0.1518 **✓works** **✓linter check pass** **✓UX is good** **✓code is good**
9	cost: $0.08 doesn't work	cost: $0.0062 doesn't work	cost: $0.01 doesn't work	cost: $0.10 doesn't work	cost: $0.12 doesn't work	cost: $0.53 **✓works** linter check fail UX issues code is bad
10	cost: $0.07 doesn't work	cost: $0.0082 doesn't work	cost: $0.02 doesn't work	cost: $0.05 doesn't work	cost: $0.16 doesn't work	cost: $0.50 doesn't work
Total	cost: $0.76 1/10 solved	**cost: $0.05** 2/10 solved	cost: $0.10 2/10 solved	cost: $0.65 3/10 solved	cost: $1.04 **5/10 solved**	cost: $3.13 **5/10 solved**
Easy solved	1/3	1/3	2/3	2/3	**3/3**	2/3
Medium solved	0/4	0/4	0/4	0/4	**1/4**	**1/4**
Harder solved	0/3	1/3	0/3	1/3	1/3	**2/3**

AIDER+O3-MINI-HIGH PROMPTING THE USER TO MANUALLY ADD THE FILE IT NEEDS



```
https://aider.chat/HISTORY.html#release-notes

> Please, solve the following issue. Title: feat: draggable splitter between the code and th
. e game screen should remember its position between the page reloads. Description: Let's pe
. rsist it in the localStorage.

Based on your repository and the issue description, the file most likely to need changes is:

 • pages/index.vue

This file probably contains the layout with the draggable splitter between the code editor
and the game screen. We'll need to update the splitter's behavior so that when its position
changes, its new value is stored in localStorage, and then on page load the saved position
is read to set the proper layout.

Please add the pages/index.vue file to the chat so I can review the relevant section and
propose SEARCH/REPLACE blocks.


Tokens: 4.3k sent, 112 received. Cost: $0.0052 message, $0.0052 session.

> pages/index.vue
```

UI ADDED BY CLAUDE CODE FOR CODE VERSIONING (ISSUE 9)



```
const threatDetectionRadius = 50;
const clusterDetectionRadius = 50; // Radius within whi
const wallPenaltyRadius = 30; // Distance from wall whe
const maxWallPenalty = 0.5; // Maximum penalty factor f
const playerSpeed = 2;
const evasionRadius = 100; // Radius within which to lo

/**
 * @param {API} api
 */
function step(api) {
    const me = api.me;
    const otherPlayers = api.otherPlayers;
    const previousPlayers = api.previousOtherPlayersSta

    const previousPlayersMap = {};
    for (const player of previousPlayers) {
        previousPlayersMap[player.username] = player;
    }

    // Initialize list for threats
    const threats = [];

    for (const player of otherPlayers) {
        if (player.username === me.username) continue;

        if (player.radius > me.radius) {
            threats.push(player); // Players larger tha
        }
    }

    // Process threats moving towards us for active eva
    let nearestThreat = null;
    let minThreatDistance = Infinity;

    for (const threat of threats) {
        const previousThreat = previousPlayersMap[threat.username];
        if (!previousThreat) continue; // Can't determine movement
```

Version from 3/21/2025, 12:16:59 PM

```
const threatDetectionRadius = 50;
const clusterDetectionRadius = 50; // R
const wallPenaltyRadius = 30; // Distan
const maxWallPenalty = 0.5; // Maximum
const playerSpeed = 2;
const evasionRadius = 100; // Radius wi

/**
 * @param {API} api
 */
function step(api) {
    const me = api.me;
    const otherPlayers = api.otherPlaye
    throw 213;
    const previousPlayers = api.previou

    const previousPlayersMap = {};
    for (const player of previousPlayer
        previousPlayersMap[player.usern
    }

    // Initialize list for threats
    const threats = [];

    for (const player of otherPlayers)
        if (player.username === me.user
```

Code history

- 3/21/2025, 12:17:06 PM
- 3/21/2025, 12:16:59 PM
- 3/21/2025, 12:16:58 PM
- 3/21/2025, 12:16:47 PM

FUTURE PLANS

- **Evaluating newer models:** We plan to test open-source QwQ-32B with Aider to see if it can match Claude's performance. If successful, this could open the door to more accessible autonomous coding (not relying on closed APIs).
- **Architect-editor agent design:** We will experiment with an "architect" mode in Aider, where one model (or one prompting style) is used to outline the solution (select files to change, perhaps write pseudo-code or steps), and another model is used as the "coder" to implement those steps.
- **Scaling to more tasks and projects:** Our current test set is small. We want to expand the evaluation to include a wider variety of issues (UI-heavy issues, algorithmic challenges, integration tasks) and on different projects (perhaps some Python backend projects, mobile app issues, etc.). This will paint a fuller picture of where autonomous LLMs excel and where they fail in software engineering.

CONCLUSION

In conclusion, state-of-the-art LLMs, when coupled with a suitable agent framework, are beginning to demonstrate practical utility in automating segments of software development in a fully unsupervised manner. They function as knowledgeable but flawed junior developers, capable of writing code and solving problems in familiar contexts, yet prone to mistakes that require oversight. By continuing to improve LLM reasoning, integrating robust self-checks, and using clever orchestrations of multiple models, we move closer to a future where AI agents can handle routine programming tasks autonomously. Such a development could significantly accelerate software engineering workflows, allowing human developers to push the boundaries of innovation with the grunt work delegated to our AI collaborators.

PRACTICAL APPLICATIONS OF STATE-OF-THE-ART LARGE LANGUAGE MODELS TO SOLVE REAL-WORLD SOFTWARE ENGINEERING PROBLEMS AUTONOMOUSLY

YURIJ MIKHALEVICH

QA WOLF

DUBAI, UNITED ARAB EMIRATES

EMAIL: YURIJ@MIKHALEVI.CH