



UC SANTA BARBARA



Carnegie Mellon University
Language
Technologies
Institute

MoatlessAI

Inference-Time Scaling of Software Agents Via Search



Antonis Antoniadēs*, Albert Örwall*
Kexun Zhang, Yuxi Xie, Anirudh Goyal, William Wang




LLM (2022-2025)

Chat Bot to Agent


LLM Models as Chat Assistants

2022:

 Give me the list of the most popular google search topics about Chat GPT 

 I am sorry, as a language model, I don't have the capability to see current or real-time Google search data. However, based on my knowledge cut off in 2021, some popular topics related to GPT-3 (which is the latest version of GPT) include:  



1. Overview and capabilities of GPT-3
2. Examples of GPT-3's natural language generation
3. GPT-3's potential uses in various industries
4. Comparison of GPT-3 to other


Chat **Model** 

✗ Interface with the internet or other tools


✗ Limited by its training data cut-off date

2025:

 Give me the list of the most popular google search topics about Chat GPT  2/2

 As of February 2025, "ChatGPT" ranked among the top 100 Google searches globally, reflecting its widespread interest. [exploringptia.com](#)
While specific data on the most popular Google search topics about ChatGPT is limited, common queries often include:

- **What is ChatGPT?** Users seek to understand the nature and purpose of ChatGPT. [youtube.com](#)
- **How to use ChatGPT?** Inquiries about accessing and interacting with the chatbot.
- **ChatGPT login** Users looking for the login page or account access information.
- **ChatGPT vs. Google Bard** Comparisons between ChatGPT and other AI chatbots, such as Google's Bard. [en.wikipedia.org](#)
- **ChatGPT API** Information on integrating ChatGPT into applications or services.
- **ChatGPT examples** Seeking sample interactions or use cases of ChatGPT.
- **ChatGPT limitations** Understanding the boundaries and challenges of using ChatGPT. [community.openai.com](#)
- **ChatGPT for SEO** Exploring how ChatGPT can assist with search engine optimization. [community.openai.com](#)

Chat **Agent** 

✓ Can selectively invoke various external tools

✓ Not limited by its training data – can always access up-to-date information

LLM Models as Chat Assistants



Give me a list of the top-performing models on SWE-Bench

As of March 10, 2025, the top-performing models on the SWE-bench leaderboard are:









1. **W&B Programmer O1 crosscheck5**: Achieved a resolution rate of 64.60% on January 17, 2025. [swebench.com](#)
2. **Blackbox AI Agent**: Achieved a resolution rate of 62.80% on January 10, 2025.
3. **CodeStory Midwit Agent + swe-search**: Achieved a resolution rate of 62.20% on December 21, 2024.
4. **Learn-by-interact**: Achieved a resolution rate of 60.20% on January 10, 2025.
5. **devlo**: Achieved a resolution rate of 58.20% on December 13, 2024.

These rankings are based on the percentage of issues resolved by each model, reflecting their effectiveness in addressing real-world software engineering problems. [openai.com](#) +7

It's important to note that the original SWE-bench dataset has been found to contain issues such as solution leakage and weak test cases, which may inflate performance metrics. To address these concerns, a refined benchmark called SWE-bench+ has been introduced, offering a more rigorous evaluation of models' capabilities. [arxiv.org](#) +5

Sources  

(This is indeed the correct order.)

Leaderboard							
Lite	Verified	Full	Multimodal				
Model		% Resolved	Org	Date	Logs	Trajs	Site
🔧 W&B Programmer O1 crosscheck5		64.60		2025-01-17	✓	✓	
🤖 Blackbox AI Agent		62.80	-	2025-01-10	✓	✓	
🧑‍🔬 CodeStory Midwit Agent + swe-search		62.20	-	2024-12-21	✓	✓	
Learn-by-interact		60.20		2025-01-10	✓	✓	
devlo		58.20		2024-12-13	✓	✓	

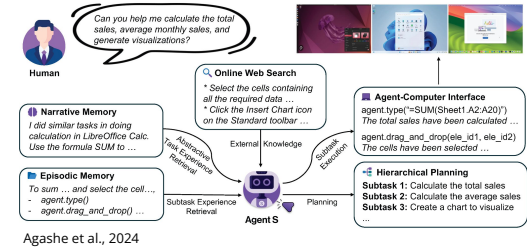
What is an LLM “Agent”

Agent: *Any system which comprises of an LLM with the ability to interface with external tools.*

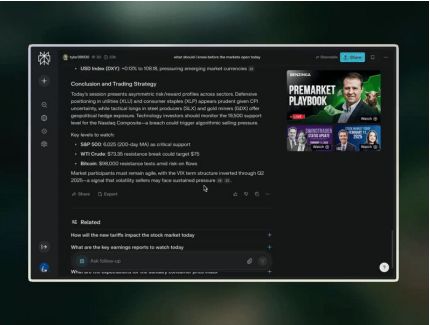
Examples of tools: Calculator, Command Line, Google Search API.

Mainstream Agents

Computer use

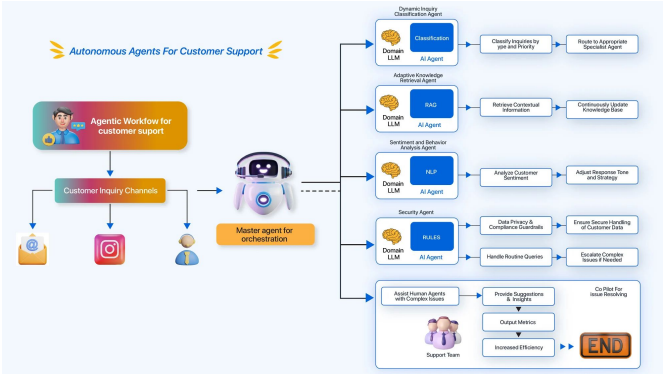


Internet Research (Perplexity Deep Research)



Perplexity

Customer Support



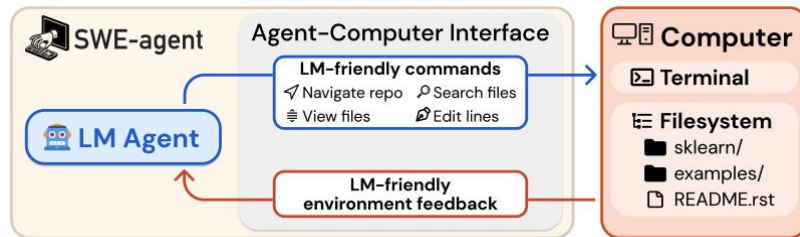
Akira AI

SWE-Agents

SWE-Agent: A system composed of a Language Model (LM) and an agent-computer interface (ACI) that enables language models to autonomously perform software engineering tasks.

SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering

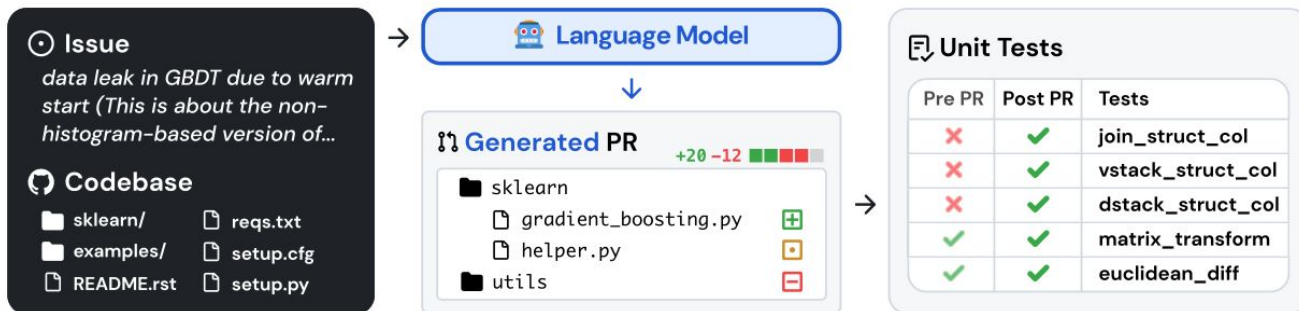
John Yang* Carlos E. Jimenez* Alexander Wettig Kilian Lieret
Shunyu Yao Karthik Narasimhan Ofir Press
Princeton Language and Intelligence, Princeton University



SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. (Yang, Jimenez, et al., 2023)

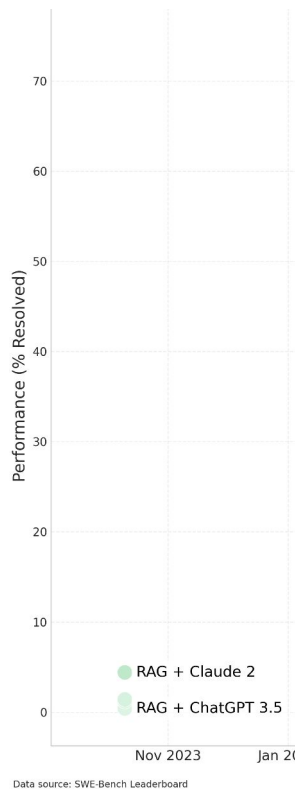
SWE-Bench

- **SWE-Bench:** An evaluation framework designed to assess the capability of language models (LMs) in resolving real-world software engineering problems. Consists of:
 - **2,294** software engineering tasks
 - Derived from actual **GitHub** issues
 - Corresponding pull requests (**PRs**)
 - Across 12 popular **Python** repositories



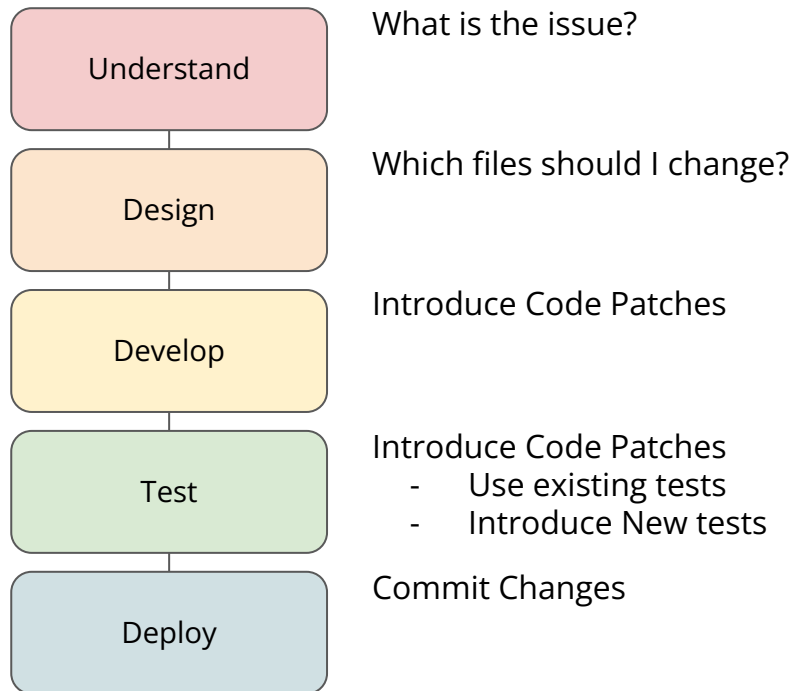
SWE-bench: Can Language Models Resolve Real-World
GitHub Issues? (Jimenez, Yang, et al., 2023)

AI Autonomous Systems

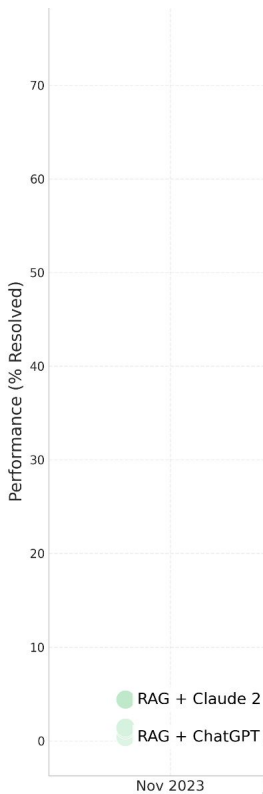


Autonomous SWE Systems Need Many Capabilities

1. **Understand** Issue
2. **Design** Solution
3. **Develop**/Iterate/Backtrack
4. **Test** final solution
5. **Deploy** Solution
6. **Maintain...**



AI Autonomous Systems are Hard



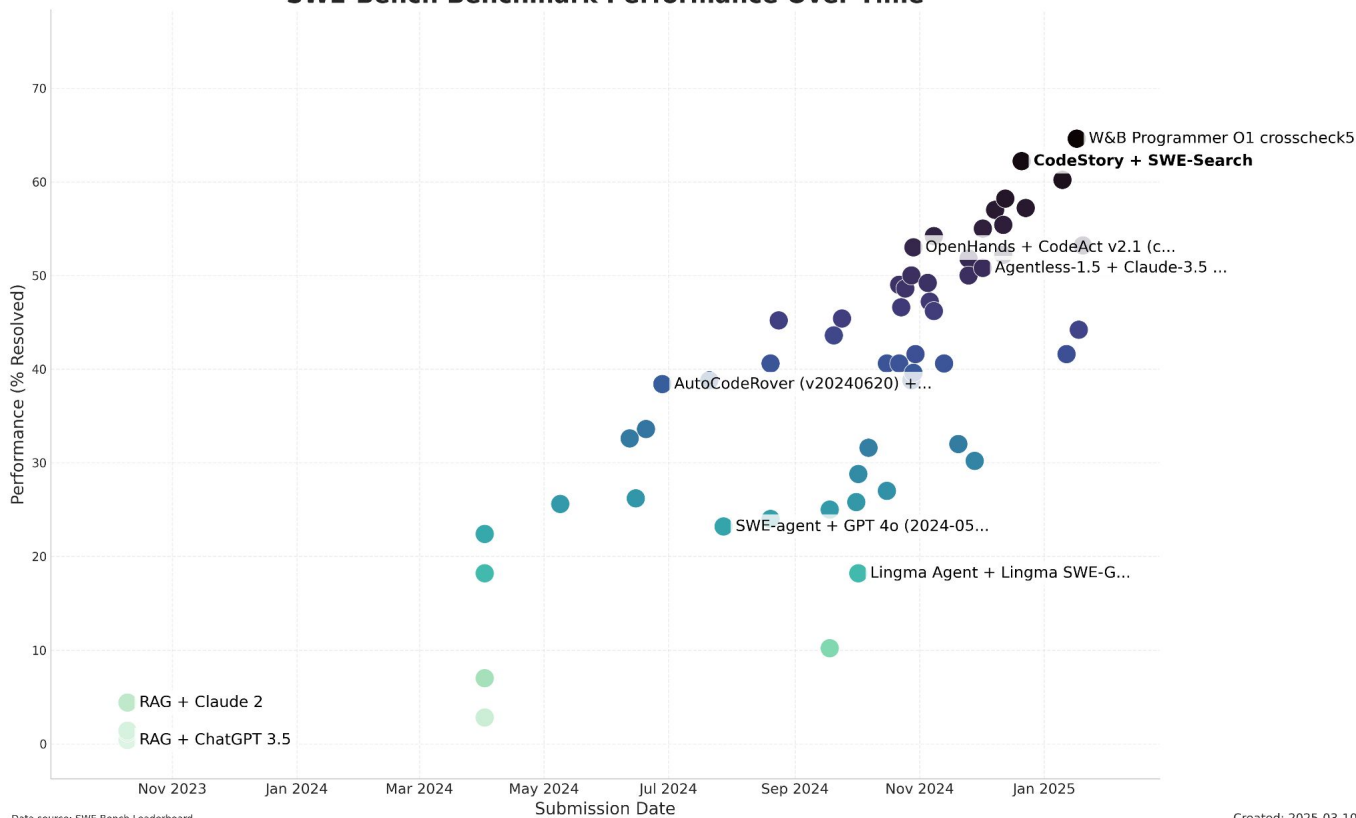
Data source: SWE-Bench Leaderboard

AC meta review:

*“Starting with such low baseline model performance might ultimately prove to be an error - **the tests may simply not be realistic to solve**, but is worth the experiment.”*

AI Autonomous Systems are Rapidly Improving

SWE-Bench Benchmark Performance Over Time



Moatless Tools

Moatless-Tools (10 mins)

- Reasoning/Motivation/Logic/Philosophy
- Tools
- Go through one example
- How to get started/join community
- Antonis/Albert: hint at issues/weaknesses of moatless and other agents, to transition to SWE-Search

Moatless Tools

- **Moatless Tools** began as a hobby project to explore how AI agents can efficiently handle coding tasks.
- **Benchmark-Driven** - Primary focus has been on testing and validating ideas using SWE-Bench, rather than developing a product.
- **Open Sharing** - Keeping the project open and accessible, with the goal of contributing knowledge and experiences back to the broader community.

Structured and systematic approach

- **Locate:** Uses abstract syntax trees, semantic vector indexing and file system operations to precisely locate code.
- **Edit:** Tackles challenges in reliable LLM-driven code editing through controlled techniques.
- **Verify:** Executes code within Docker containers, parsing test results into concise LLM feedback.

Demo

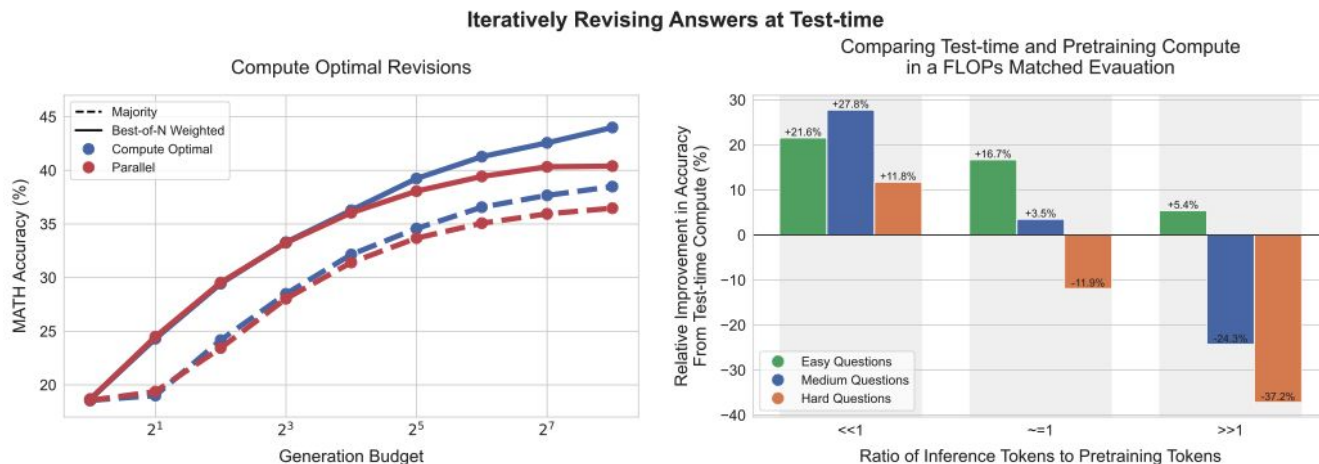
SWE-Search

Inference-time Scaling

- How to improve a model's performance, *without any additional training*.

Inference-time Scaling

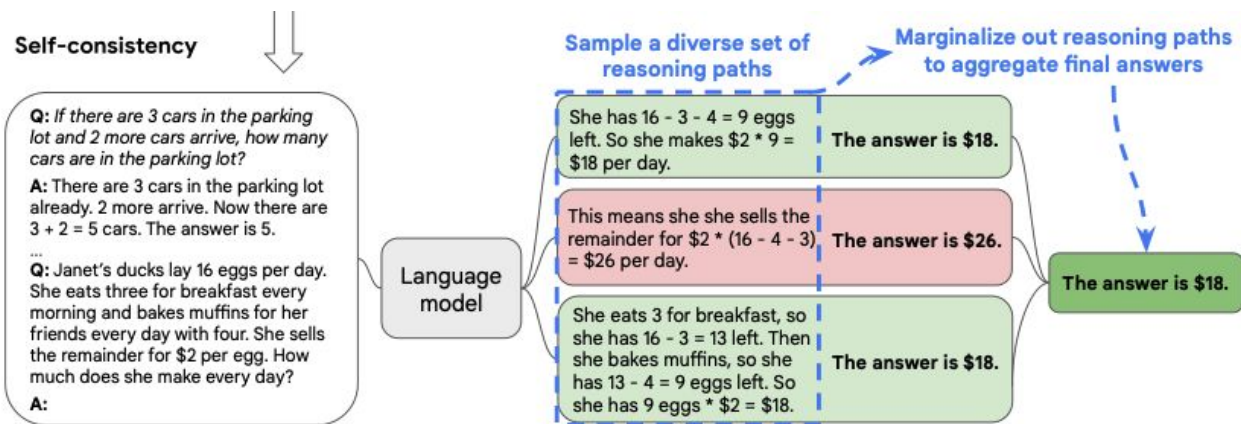
- How to improve a model's performance, *without any additional training*.



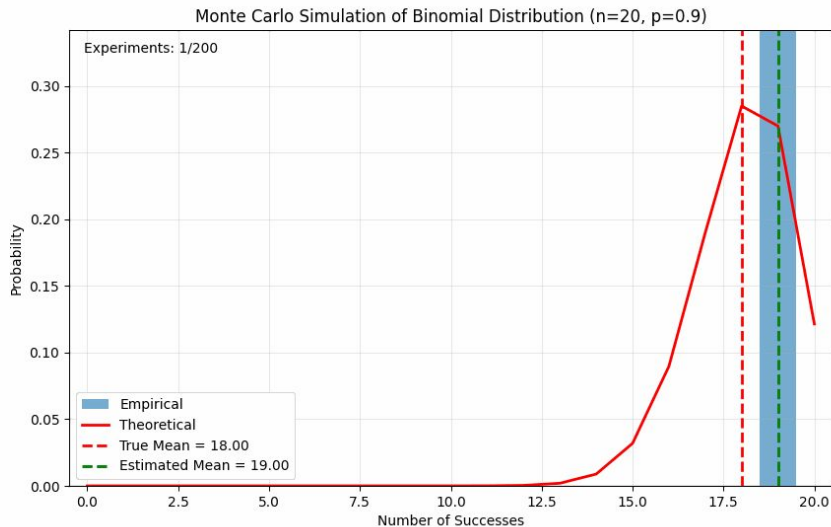
Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters (Snell et al., 2024)

Majority Voting

- Sample LLM multiple times and choose most common answer.



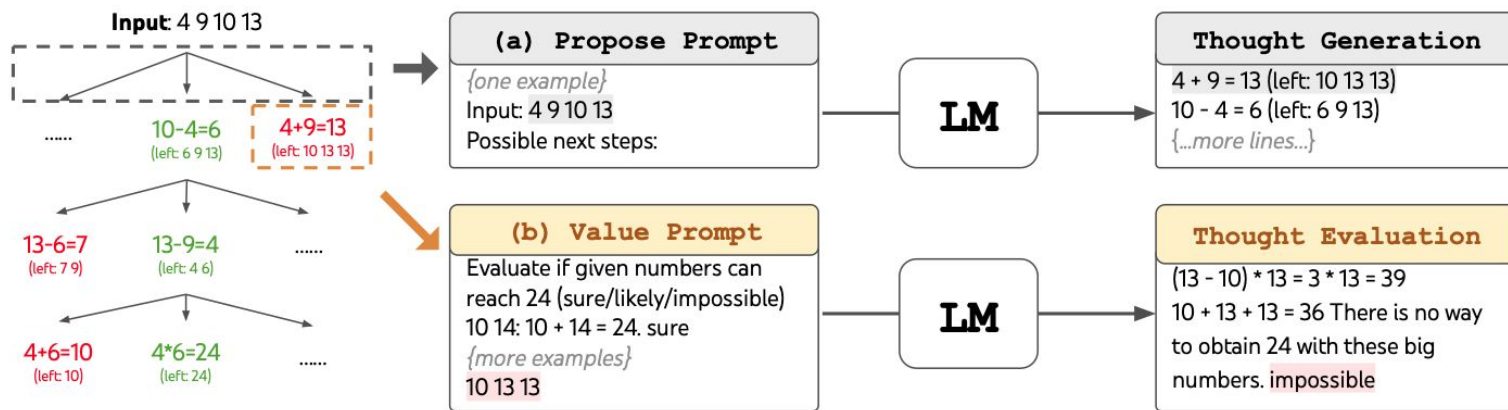
Majority Voting is a Monte-Carlo Simulation



Note: Only works if the $p(\text{correct}) > p(\text{wrong})$

Tree of Thoughts

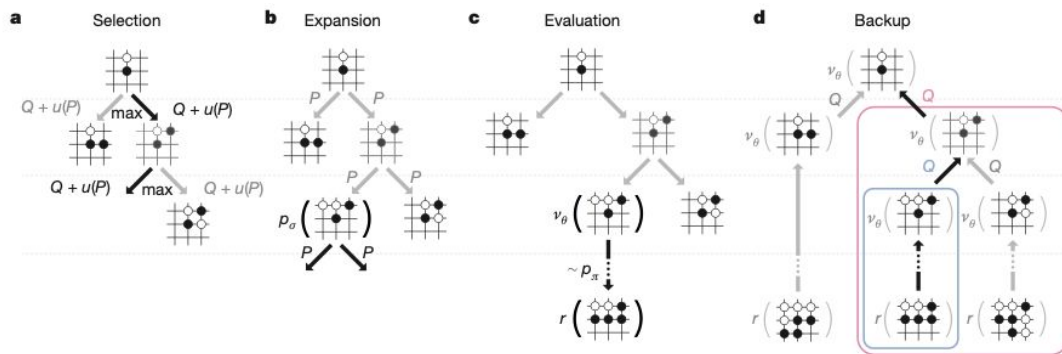
- Model the reasoning process as a tree.
 - Sample LLM to expand multiple trajectories
 - Use LLM to self-evaluate reasoning states.



Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Yao et al., 2023)

Tree Search, Monte-Carlo Simulations and ML

- Deepmind **Alpha**-(chess, zero, go etc...) line of models are based on:
 - **Monte-Carlo Simulation**
 - **Tree Search**
- Called **MCTS**



Mastering the game of Go with deep neural networks and tree search (Silver et al., 2016)

- **Policy Network (π):**
 - Generates Action
- **Value Network (v):**
 - Evaluates State

Inference Scaling... and Agents?

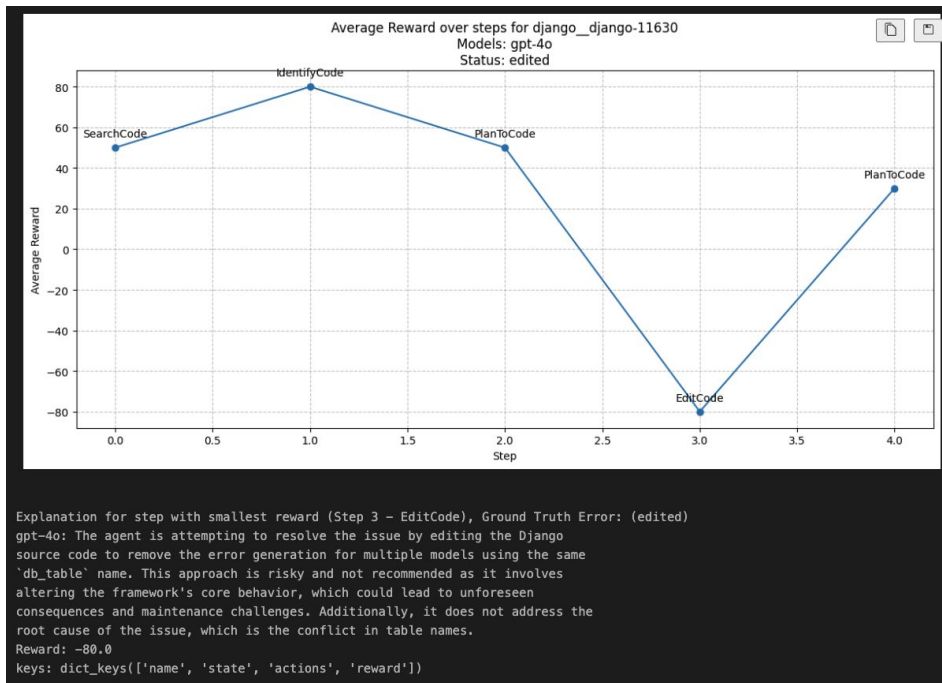
- How to enable the scaling of performance on Agents (using SWE-Agents as case study)
 - Challenges:
 - Complex Environment
 - Many different “states” (plan, edit, test – how do we evaluate each one?)
 - How do we save previous states for effective backtracking?
 - How to make this fast?

SWE-Search: Inference-time scaling of SWE-Agents

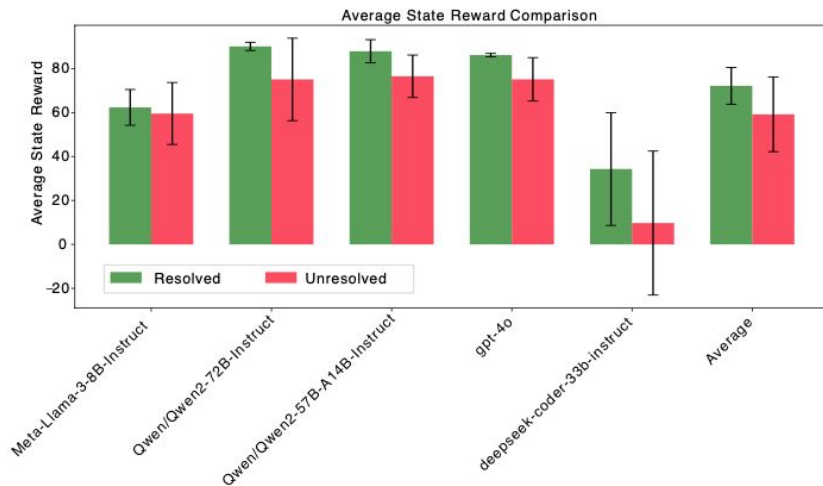
- Key Insight:
 - The flexibility of Language Models allows them to self-evaluate on a vast variety of states/actions. As long as we provide some reasonable **guidance** for each type!
 - LLMs can therefore act both as a **Policy** and **Value Network**

Ability of models to evaluate states/actions/trajectories

State Rewards vs. Failure Mode (=Edit)



Av. State Rewards vs. Final Outcome



Not only can LLM **quantitatively** evaluate States, but it can also provide **qualitative explanations**

Evaluating State Utility

- State Value: The ability of a state to reach a final solution that resolves the issue.

Value Function Prompts

Plan

- Evaluate correctness, mistakes, and test results.
- Ensure meaningful progress, avoid repetition.
- Score (-100 to 100) based on completeness and efficiency.

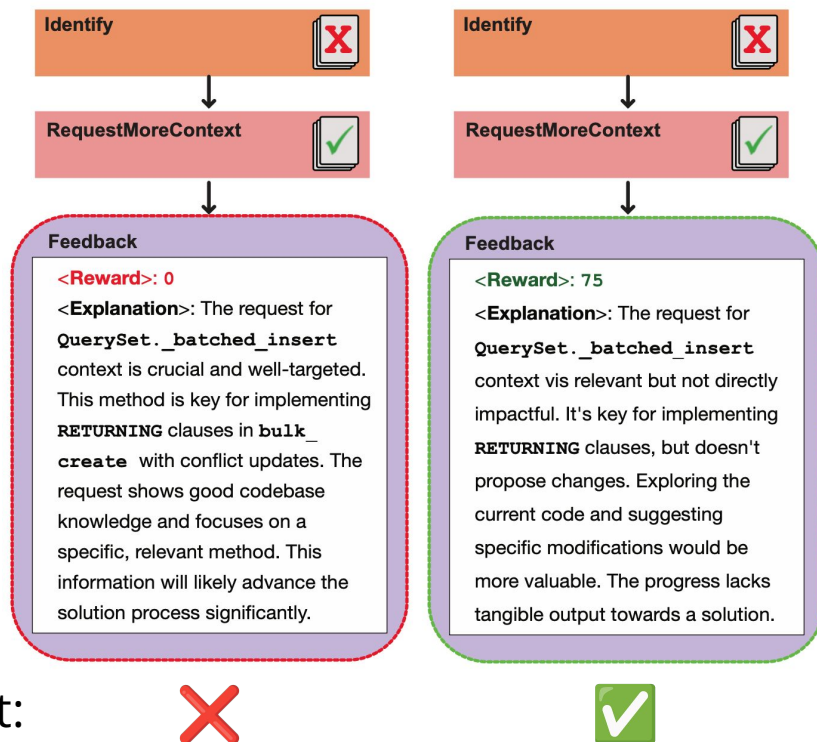
Search

- Assess relevance, necessity, and specificity of requests.
- Ensure efficiency and problem-solving progress.
- Score (-100 to 100) based on accuracy and need.

Edit

- Verify correctness, optimization, and test coverage.
- Penalize missing tests or unnecessary changes.
- Score (-100 to 100) based on solution quality.

Prompting is crucial for correct evaluation

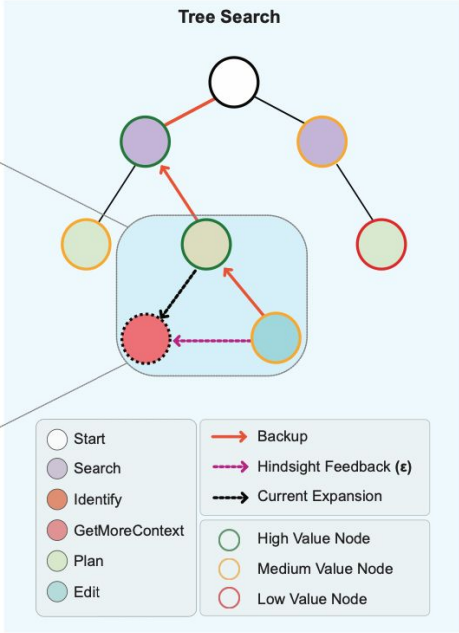


State-Specific prompt:

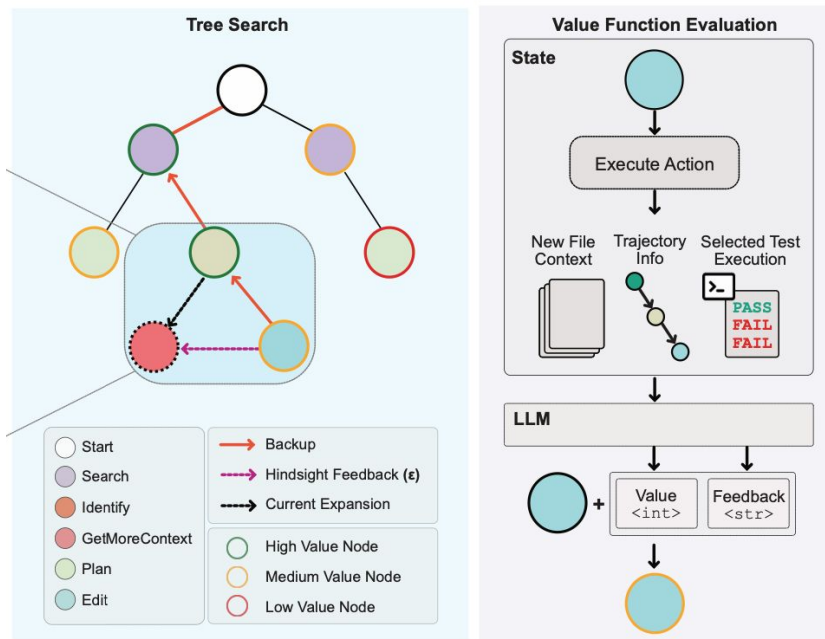
SWE-Search Building Blocks

- **Expander** - Generates and adds new nodes to the tree
- **Selector** - Selects the next node in the tree to execute
- **FeedbackGenerator** - Generates feedback for the agent based on outcomes from previously executed nodes
- **Agent** - Core component responsible for executing actions
- **ValueFunction** - Evaluates the utility of each state/action
- **Discriminator** - Identifies and selects the final solution from multiple candidates

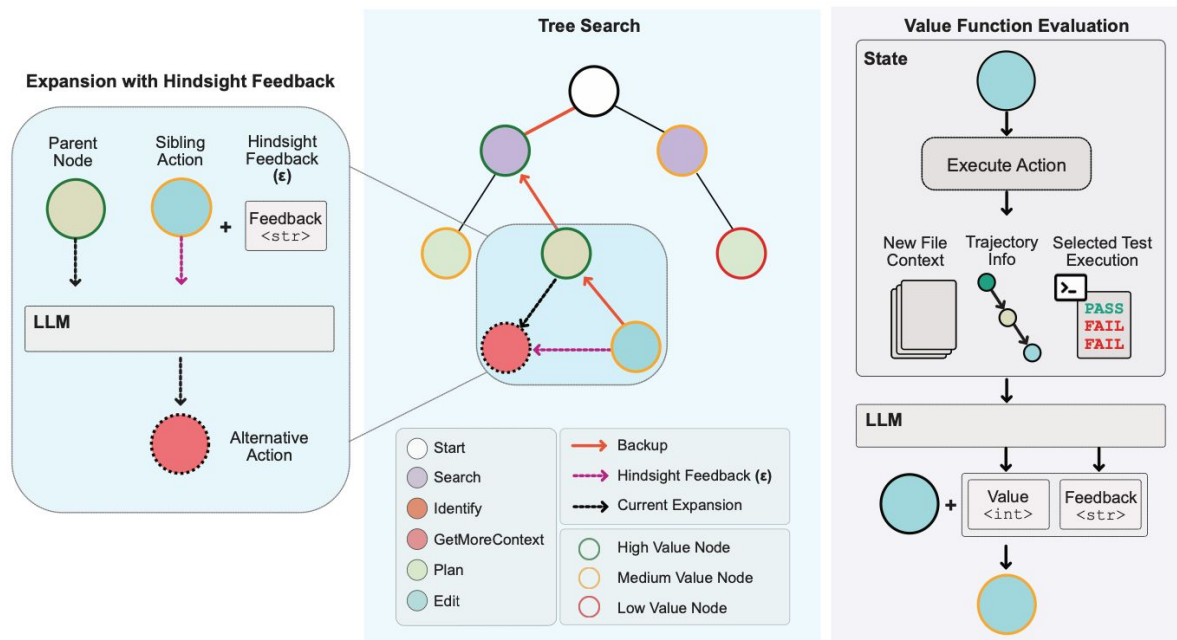
Search Tree



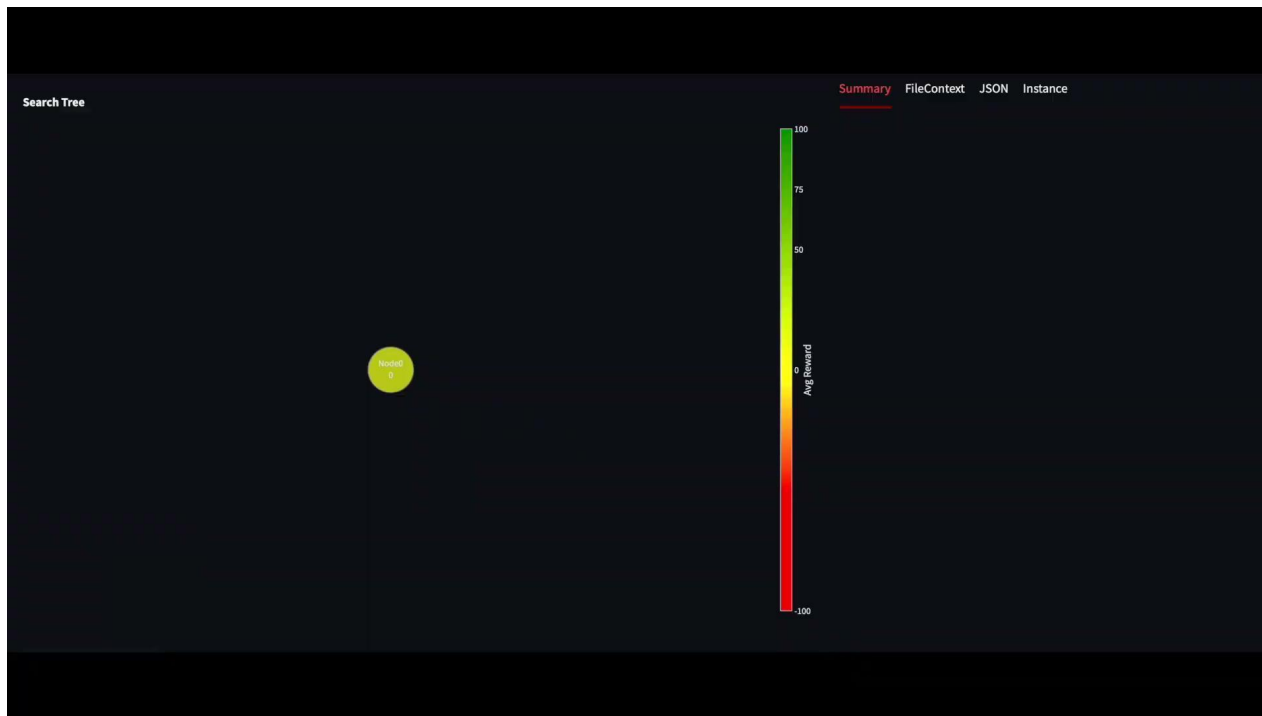
Search Tree + Value Function



Search Tree + Value Function + Self-Refinement

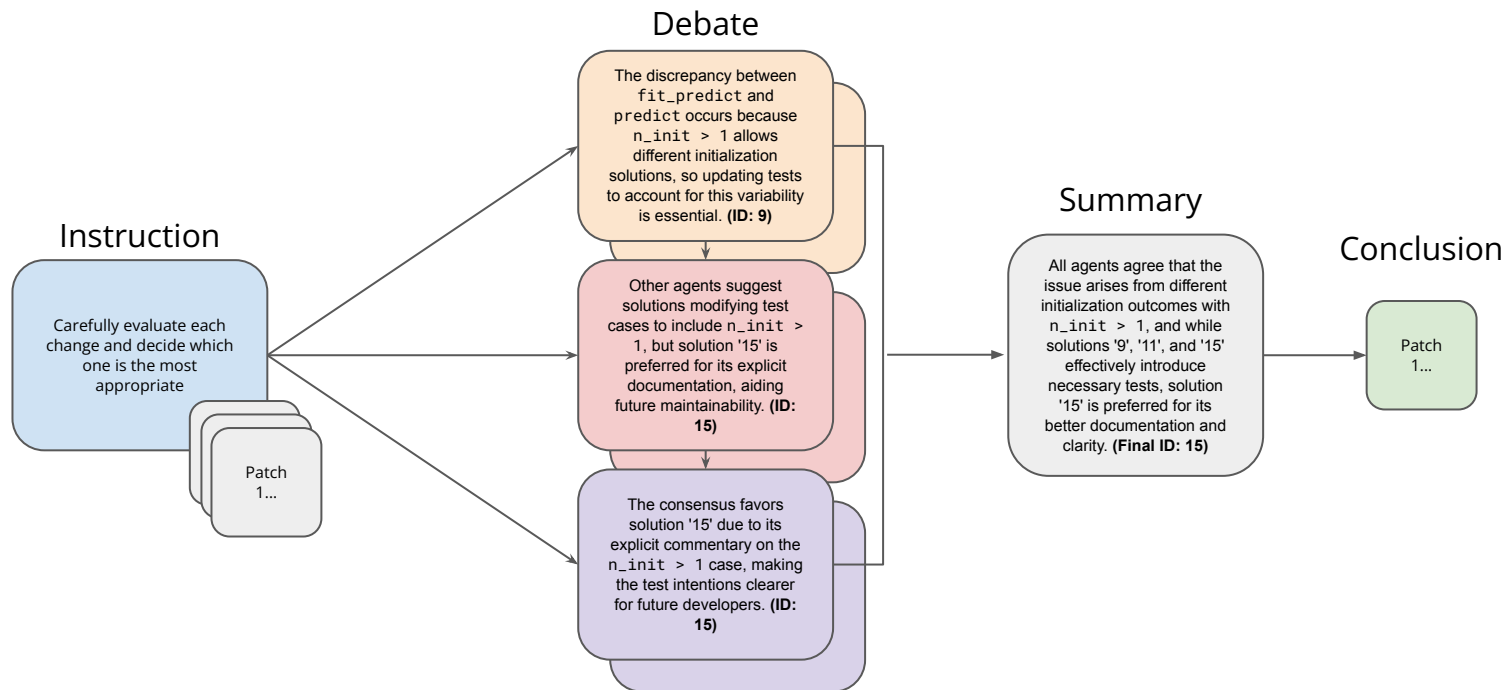


Demo

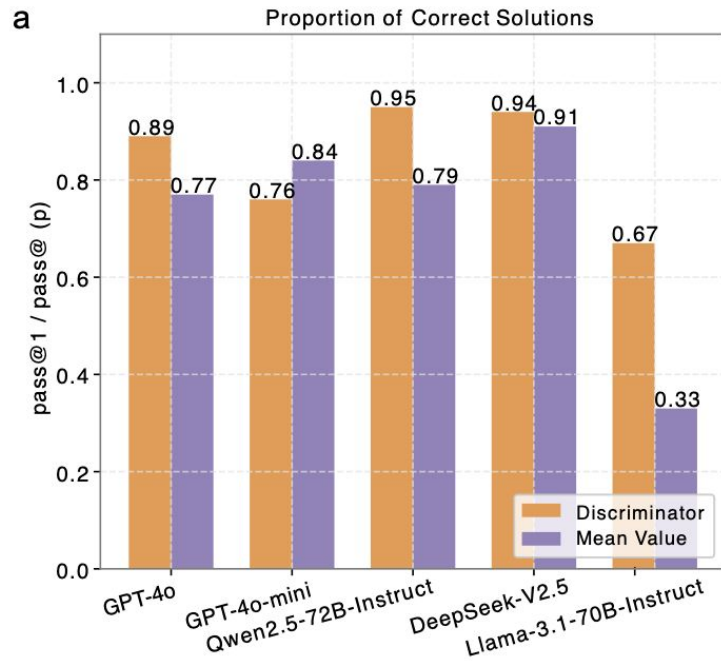


Multi-Agent Debate as Verifier (*Discriminator*)

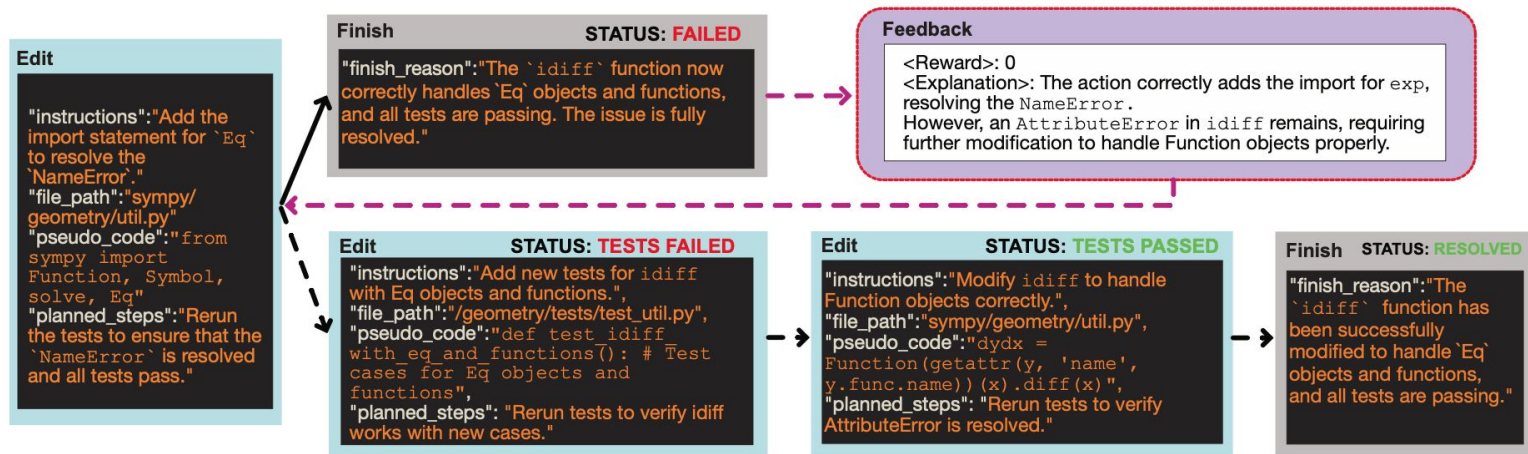
- Majority voting doesn't work -> **p(correct) is often \ll than p(wrong)**
 - Instead, utilize **multi-agent** debate



Training-free Multi-Agent Debate Improves Performance

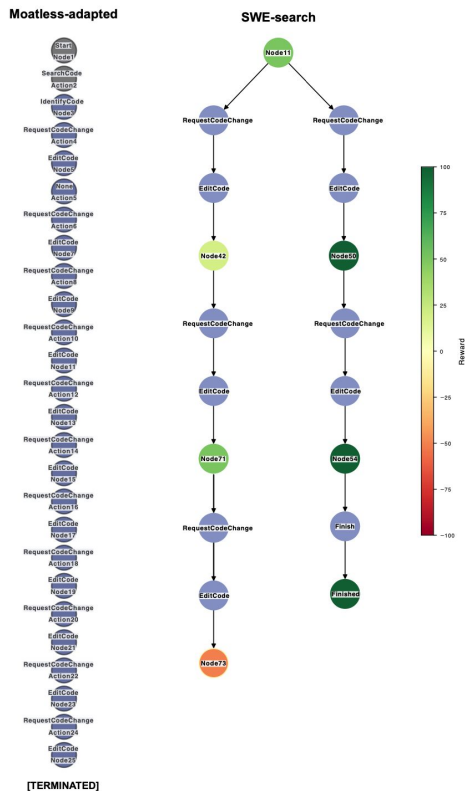


Example: Discovering missing tests



Example: Getting out of Repetitive Dead-ends

Agent is stuck
running wrong,
failing tests



Value function Detects
failing tests, and
re-expands with feedback
to avoid repeating the
same outcome

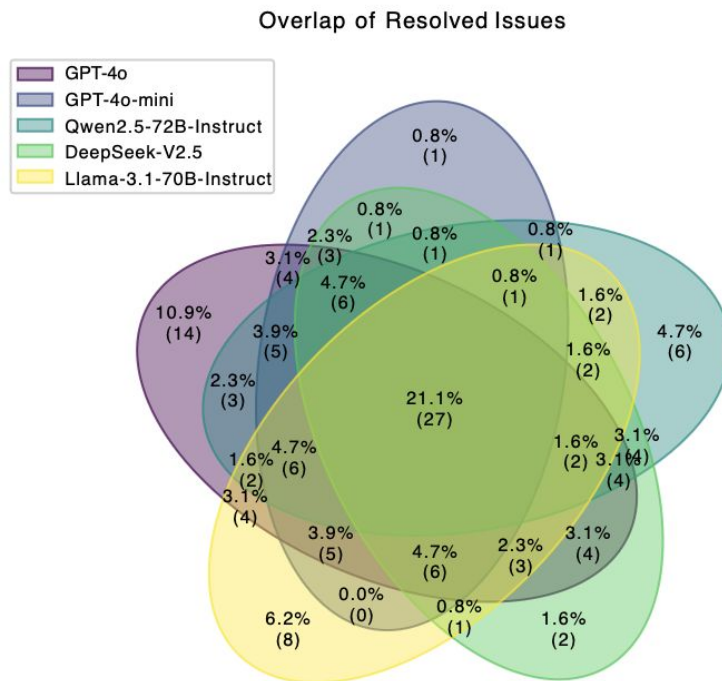
20% boost in performance w/o training

Table 1: Resolve Rate Comparison, SWE-bench Lite

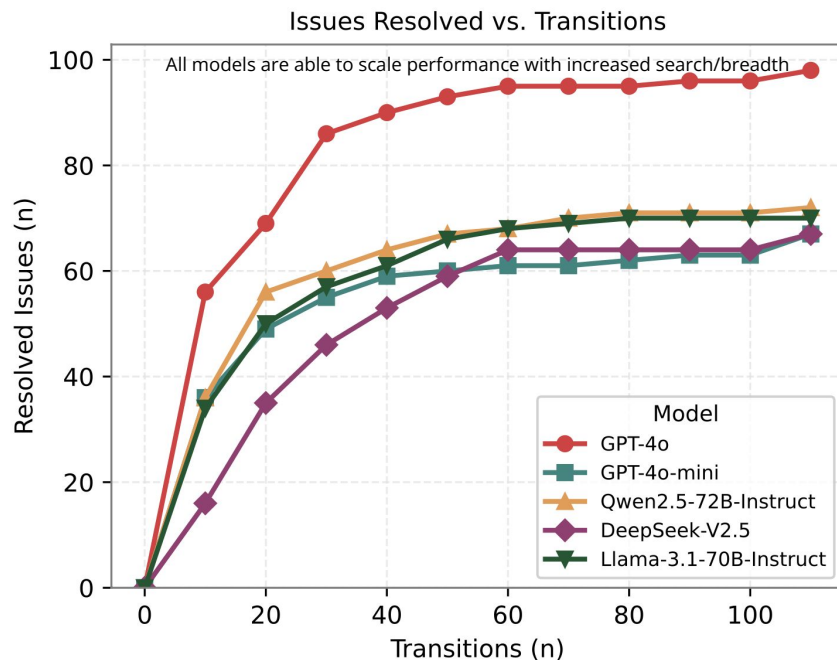
Model	Moatless-v0.0.2	Moatless-Adapted	SWE-Search	% Δ
GPT-4o	24.3	25.7	31.0	+17
GPT-4o-mini	–	13.0	17.0	+24
Qwen-2.5-72b-Instruct	–	18.0	24.7	+27
Deepseek-V2.5	–	16.3	21.0	+22
Llama-3.1-70b-Instruct	–	13.6	17.7	+23
Mean % Δ				+23

Different Models Resolve Non-overlapping Issue Subsets

Surprisingly, even the lowest-performing model (Llama-3.1-70b, could solve 8 instances that were not resolved by any other model

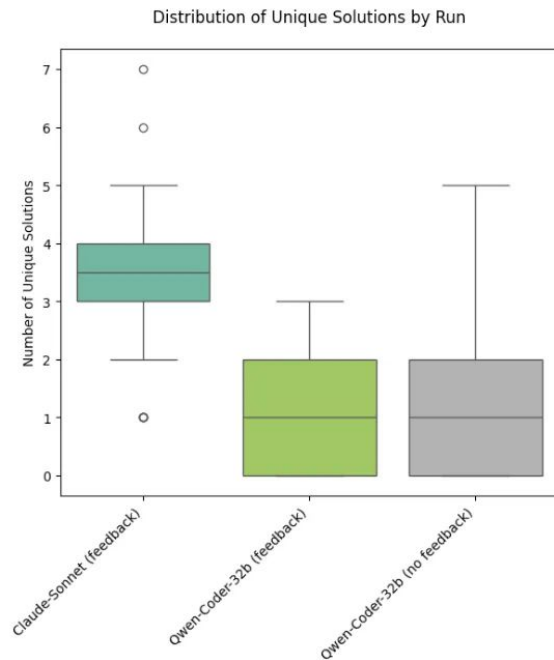
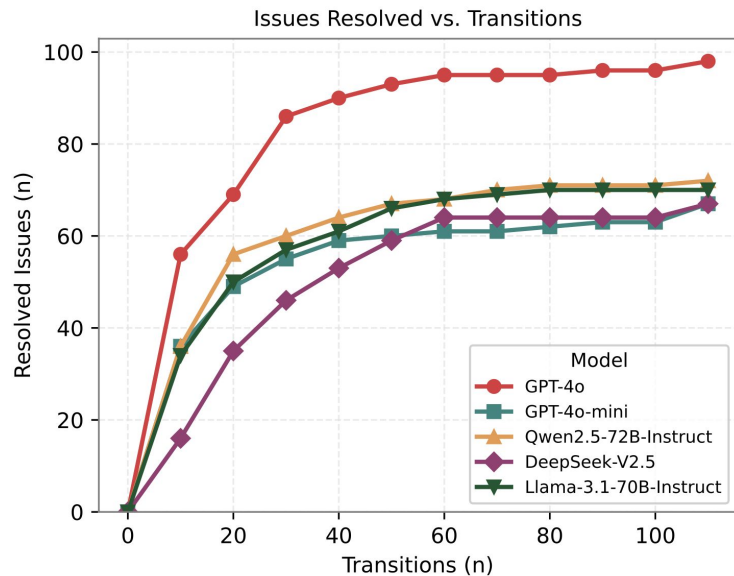


Inference-time Scaling for Agents



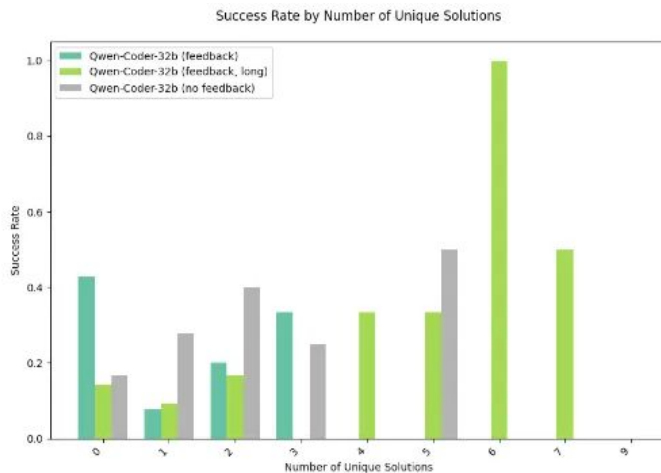
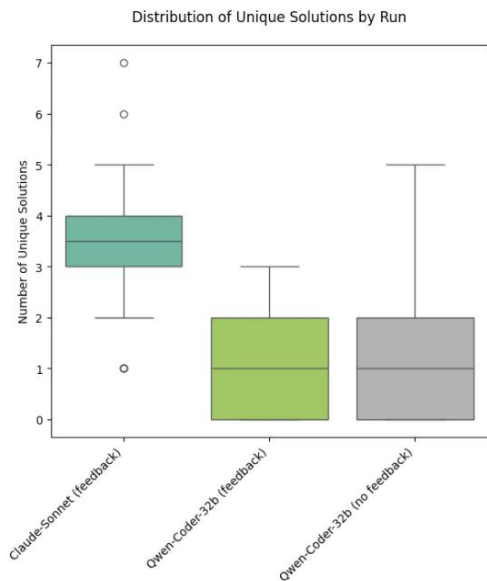
Future

Why does search scaling plateau?



Models have a limited capacity to generate unique solutions, *even given feedback*.

Why does search scaling plateau? - *Unique Solutions*



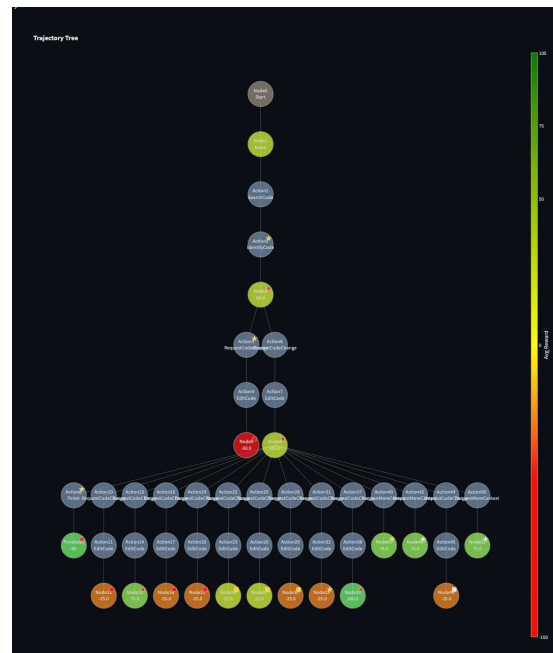
Models have a limited capacity to generate unique solutions, *even given feedback*.

Why does search scaling plateau? - *Self-Planning*

Provided Ascii Tree

```
MCTS Tree (PROVIDING FEEDBACK TO MODEL)
└─[0] Pending (depth: 0, visits: 31, value: 0.00, avg: 71.13)
  └─[1] SearchCode (depth: 1, visits: 31, value: 0.00, avg: 71.13)
    └─[3] PlanToCode (depth: 2, visits: 31, value: 60.00, avg: 71.13)
      Feedback: Consider searching for existing implementations of `__repr__` in other
      classes that handle `functools.partial`, or look for utility functions that
      unwrap partial functions. This could provide insights or reusable code that can
      be integrated into the `__repr__` method of `ResolverMatch` to meet the
      requirements of the problem statement more effectively.
      └─[4] PlanToCode (depth: 3, visits: 30, value: 0.00, avg: 71.22)
        Explanation: <Explanation>: The executed action involved searching for the
        `__repr__` function in the `ResolverMatch` class, which is a logical step in
        addressing the problem of how `functools.partial` objects are represented.
        However, the action did not include any modifications to the `__repr__` method
        itself, which is necessary to handle `functools.partial` correctly. The problem
        statement suggests unwrapping `functools.partial` objects in the `__repr__`, but
        the agent only searched for the function without implementing any changes. This
        indicates a lack of progress towards solving the original issue. To improve
        the solution, the agent should modify the `__repr__` method to check if
        `self.func` is an instance of `functools.partial` and, if so, extract the
        underlying function and any arguments provided. This would enhance the output of
        the `__repr__` method, making it more informative and aligned with the
        requirements outlined in the problem statement. The tests have passed, but this
        does not reflect the completeness of the solution since the core issue remains
        unaddressed. <Reward>: 0
        └─[6] PlanToCode (depth: 4, visits: 29, value: 100.00, avg: 71.77)
          Feedback: <Explanation>: The proposed changes to the `__repr__` method in the
          `ResolverMatch` class are appropriate and effectively address the problem of
          handling `functools.partial` objects. The agent correctly identifies when the
          `func` attribute is an instance of `functools.partial` and constructs a more
          informative string representation that includes the underlying function's module
          and name, along with its arguments and keywords. This change enhances the
          clarity of the output when debugging or logging, making it easier to understand
          the state of the `ResolverMatch` instance. The Git diff accurately reflects the
          intended modifications, with no unintended changes or formatting issues. The
          changes are confined to the `__repr__` method, and the overall structure of the
          class remains intact. Additionally, the test results indicate that all tests
          passed successfully, confirming that the modification did not introduce any
          regressions or errors. The solution is well-aligned with Pythonic principles,
          utilizing f-strings for string formatting, which is both readable and efficient.
          <Reward>: 100
```

Expanded Tree



Models have a limited capacity to **strategically plan** and **allocate resources**.

The new meta: Training for inference-time Scaling

- Models need to be trained to scale better at inference-time

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a + x}} = x$ is equal to

Response: <think>

To solve the equation $\sqrt{a - \sqrt{a + x}} = x$, let's start by squaring both ...

$$\left(\sqrt{a - \sqrt{a + x}}\right)^2 = x^2 \implies a - \sqrt{a + x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...

Wait, wait. Wait. That's an aha moment I can flag here.

Let's reevaluate this step-by-step to identify if the correct sum can be ...

We started with the equation:

$$\sqrt{a - \sqrt{a + x}} = x$$

First, let's square both sides:

$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

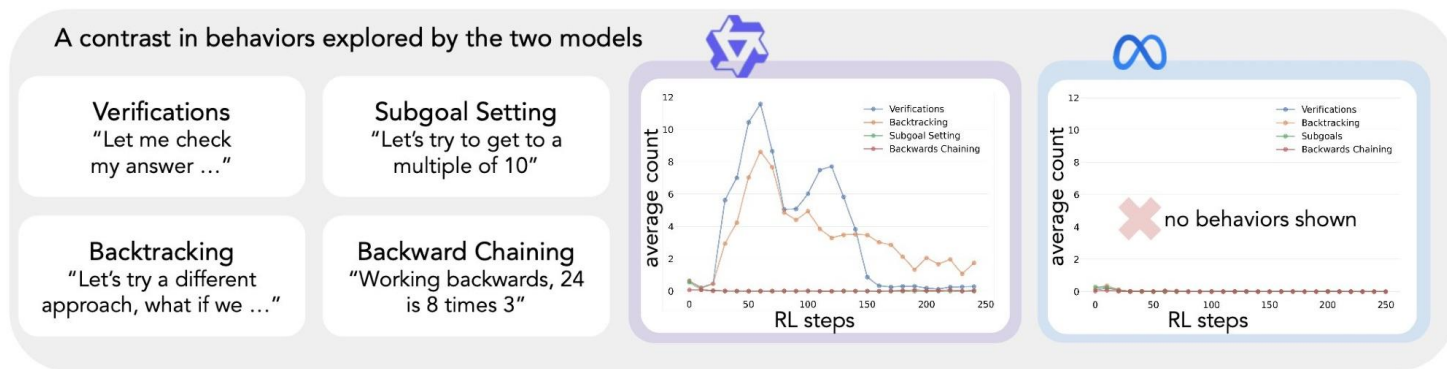
Next, I could square both sides again, treating the equation: ...

...

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning,
(Deepseek AI, 2025)

Not all models are created equal

- The abilities of models to self-improve, are dependent on a strong foundation (base model)



Cognitive Behaviors that Enable Self-Improving Reasoners, or, Four Habits of Highly Effective STaRs (Gandhi et al., 2025)

Future

- Training models to search
- More favorable inference-time scaling
 - Can we be better than log-linear?
- Fully autonomous, multi-agent software systems
- Self-improving, Self-assembling systems

How to get started

- Use Moatless Tools as a library: <https://github.com/aorwall/moatless-tools>
- SWE-Search components: <https://github.com/aorwall/moatless-tree-search>
- Docker container solution with a UI currently in development.
- For assistance or pilot-testing the UI join our Discord:
<https://discord.gg/74VX8ppBEg>

Thanks!

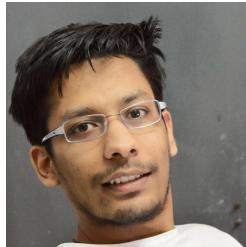
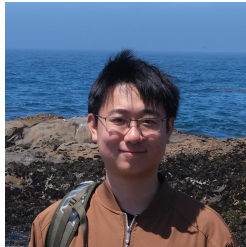
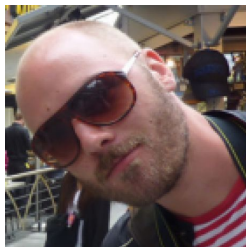


UC SANTA BARBARA



Carnegie Mellon University
Language
Technologies
Institute

MoatlessAI

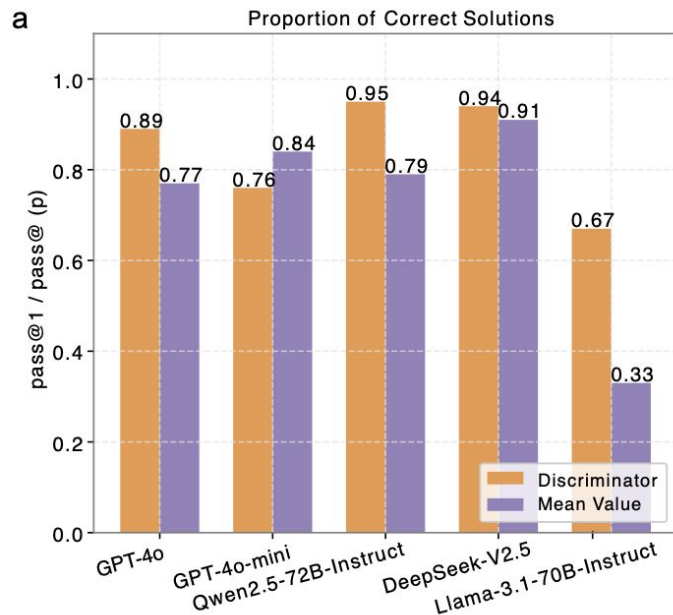


Github: <https://github.com/aorwall/moatless-tree-search>

X:
@a-antoniades
@aorwall

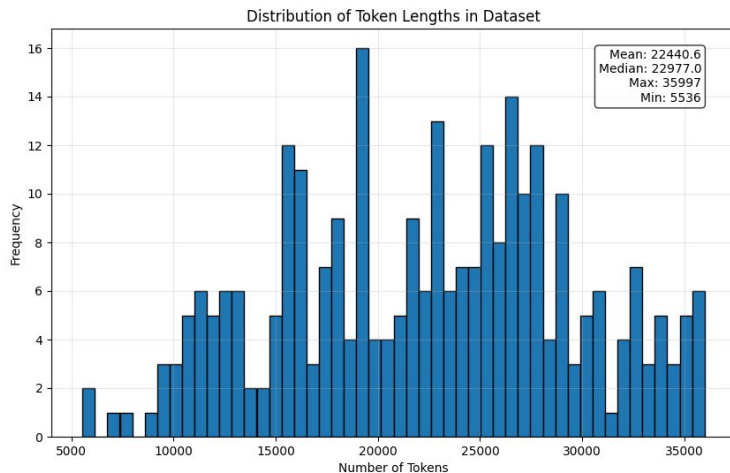
Auxiliary Slides

Training-free Multi-Agent Debate Improves Performance

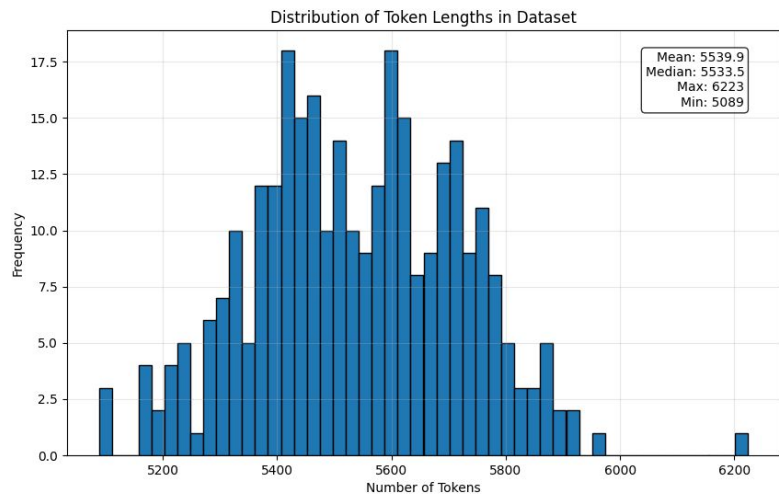
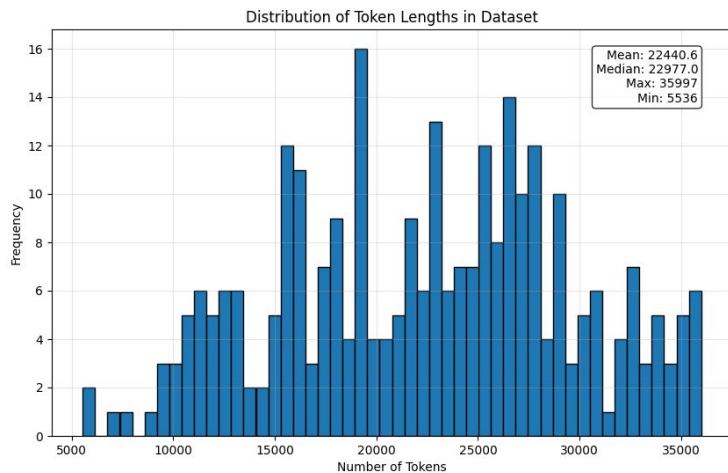


Context Bottleneck

- Instances are very long
- Attention scales quadratically
- Need at least 8b param. Model
- Working on efficiency improvements
 - I have been able to now fit 8b model on 1 40 GB A100 GPU
 - Working to integrate with my whole pipeline

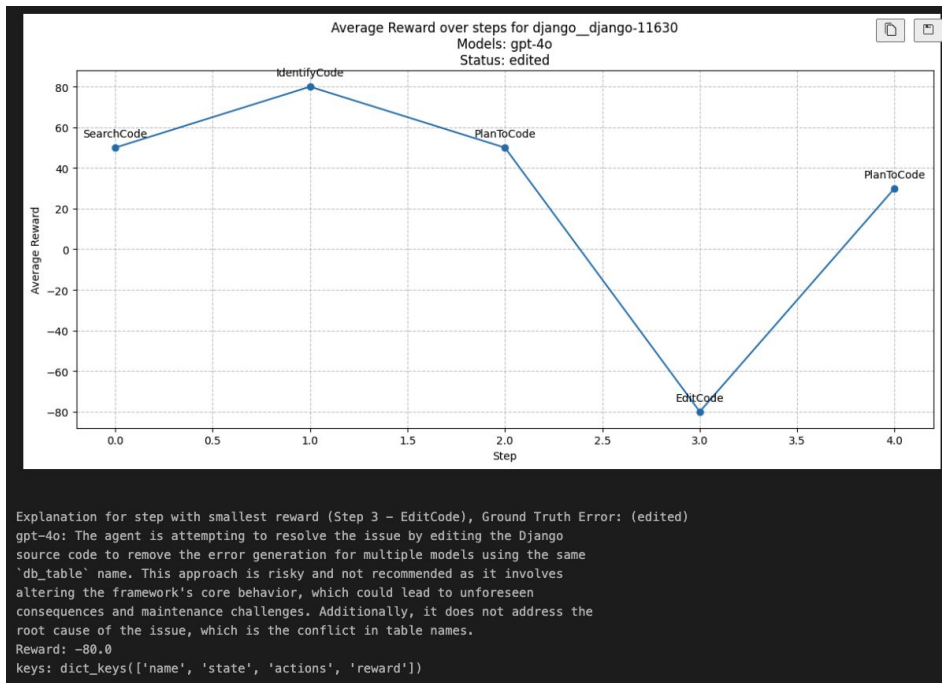


Shorten Context using Smart Window + Summarization

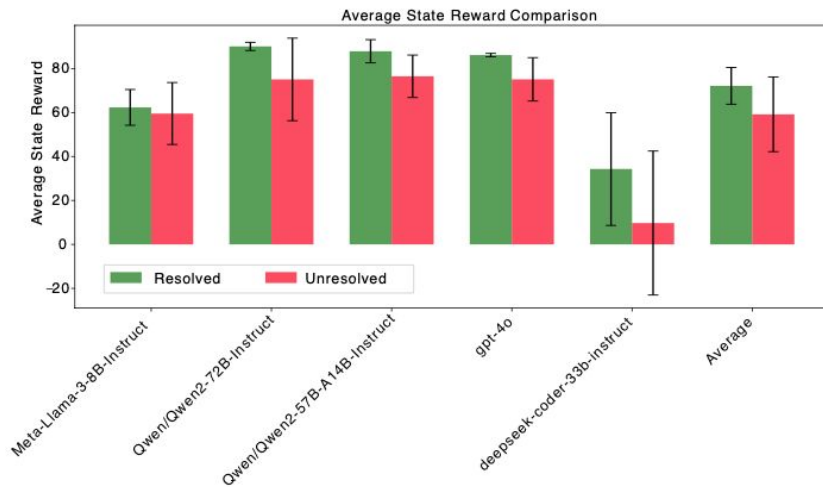


Ability of models to evaluate states/actions/trajectories

State Rewards vs. Failure Mode (=Edit)



Av. State Rewards vs. Final Outcome



Not only can LLM **quantitatively** evaluate States, but it can also provide **qualitative explanations**