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

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

John Yang[1,2*] **Carlos Jimenez**[1,2*] **Alexander Wettig**[1,2] **Kilian Lieret**[1,2]
Shunyu Yao[1,2] **Karthik Narasimhan**[1,2] **Ofir Press**[1,2]
[1]Princeton University [2]Princeton Language and Intelligence
{jy1682,carlosej,awettig,kl5675,shunyuy,karthikn,ofirp}@princeton.edu

Presenter: Yu (Hope) Hou
CMSC 818I 09/10

Background: Why Software Engineering Task

Real-world software engineering is rich and more challenging for LLMs;

SWE-bench: Evaluation framework consisting of 2,294 software engineering problems

GitHub issues and corresponding pull requests; 12 popular Python repositories

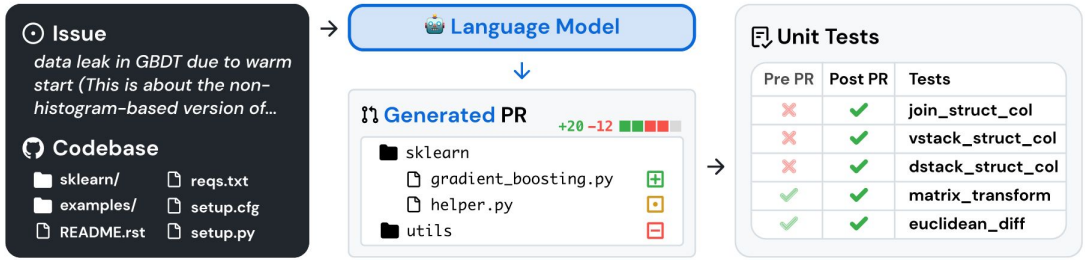


Figure 1: SWE-bench sources task instances from real-world Python repositories by connecting GitHub issues to merged pull request solutions that resolve related tests. Provided with the issue text and a codebase snapshot, models generate a patch that is evaluated against real tests.

https://www.swebench.com/

Leaderboard

Lite | Verified | Full

Model	% Resolved	Date	Logs	Trajs	Site
CodeStory Aide + Mixed Models	43.00	2024-07-02	🔗	-	🔗
Honeycomb	38.33	2024-08-20	🔗	🔗	🔗
AbanteAI MentatBot + GPT 4o (2024-05-13)	38.00	2024-06-27	🔗	-	🔗
Gru(2024-08-11)	35.67	2024-08-11	🔗	🔗	🔗
Isoform	35.00	2024-08-29	🔗	🔗	🔗
SuperCoder2.0	34.00	2024-08-06	🔗	🔗	🔗
	34.00	2024-07-23	🔗	-	🔗
	33.00	2024-06-22	🔗	🔗	🔗
	31.33	2024-06-17	🔗	-	🔗
3)	30.67	2024-06-21	🔗	🔗	🔗
	29.67	2024-07-21	🔗	🔗	🔗
	29.67	2024-08-08	🔗	🔗	🔗
	28.33	2024-06-04	🔗	-	🔗
	28.00	2024-06-12	🔗	-	🔗
	27.67	2024-07-06	🔗	🔗	🔗
	27.33	2024-06-30	🔗	-	🔗
	26.67	2024-06-23	🔗	🔗	🔗
OpenDevin + CodeAct v1.8	26.67	2024-07-25	🔗	🔗	🔗
IBM Research Agent-101	26.67	2024-06-12	🔗	-	🔗
Aider + GPT 4o & Claude 3 Opus	26.33	2024-05-23	🔗	-	🔗
Moatless Tools + GPT 4o (2024-05-13)	24.67	2024-06-17	🔗	🔗	🔗
OpenCSG StarShip CodeGenAgent + GPT 4 (0613)	23.67	2024-05-24	🔗	-	🔗
SWE-agent + Claude 3.5 Sonnet	23.00	2024-06-20	🔗	🔗	-

All Hands AI @allhands_ai · Sep 5

We are proud to announce that All Hands has raised $5M to build the world's best software development agents, and do it in the open 🙌

all-hands.dev

Thank you to @MenloVentures and our wonderful slate of investors for believing in the mission!

> **TechCrunch** @TechCrunch · Sep 5
>
> All Hands AI raises $5M to build open source agents for developers
> tcrn.ch/3MxNUvB

Paper we will discuss today!

SWE-agent: Overview

"LM acts as an agent when it interacts with an environment by iteratively taking actions and receiving feedback ..."

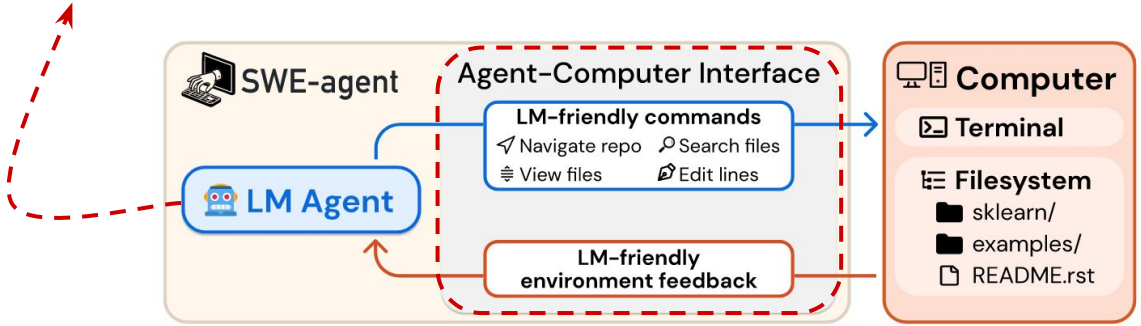


Figure 1: SWE-agent is an LM interacting with a computer through an agent-computer interface (ACI), which includes the commands the agent uses and the format of the feedback from the computer.

Agent-Computer Interface (ACI): Motivation

The **interface** LM agents use to interact with computers;

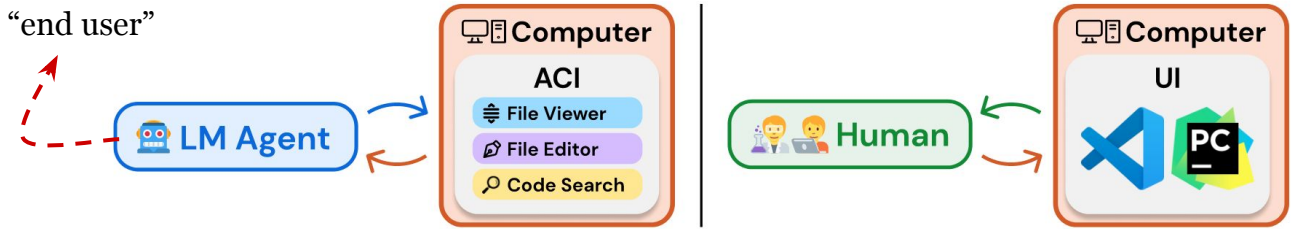
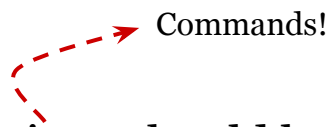


Figure 2: Specialized applications like IDEs (e.g., VSCode, PyCharm) make scientists and software engineers more efficient and effective at computer tasks. Similarly, ACI design aims to create a suitable interface that makes LM agents more effective at digital work such as software engineering.

Agent-Computer Interface (ACI): Design Principles

(… draw from HCI)



1. Actions should be **simple and easy** to understand for agents
2. Actions should be **compact and efficient**
3. **Environment feedback** should be informative but concise
4. Guardrails **mitigate error** propagation and hasten recovery

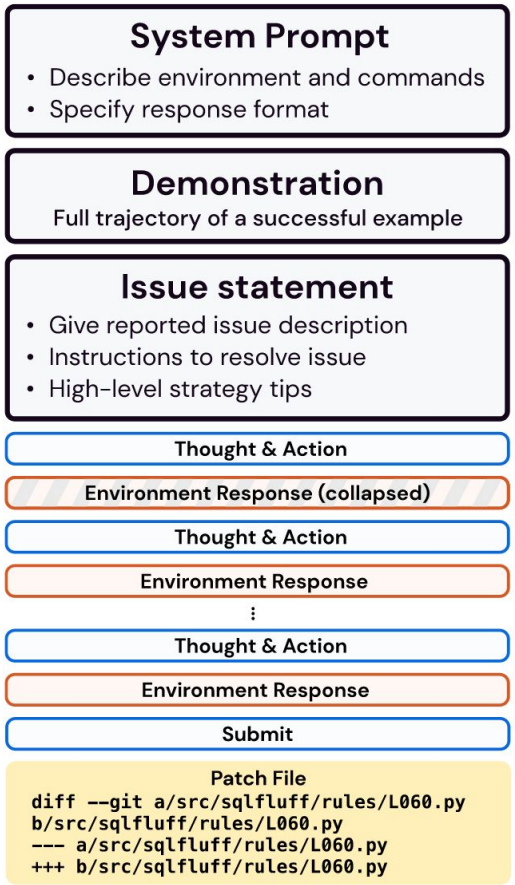
SWE-agent: Design

To solve software engineering problems:

1/ Localization: Identify file(s)/line(s) causing the issue.

2/ Editing: Generate fixes addressing the given issue.

3/ Testing: Write new scripts or modify existing test files to reproduce the issue and/or verify if fixes are correct.



SWE-agent: Prompt Workflow

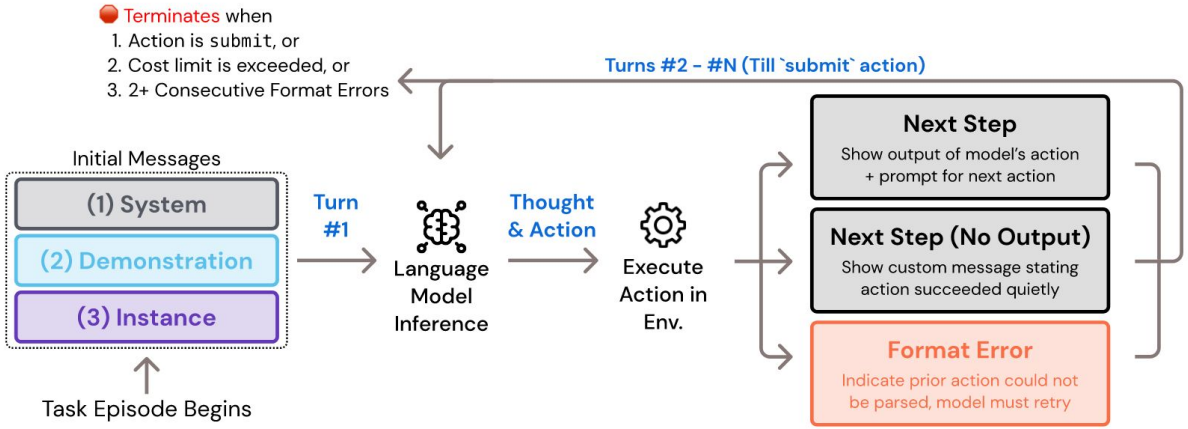


Figure 26: The flow of prompt templates throughout a single SWE-agent task instance episode. The system, demonstration, and issue templates are shown all together at the beginning of the task episode, followed by turn-specific prompts that are shown depending on whether the agent response is well-formatted and whether the action has standard output.

SWE-agent*: ACI Components

1. Search and navigation

Find Files

```
find_file atmosphere.py
```

```
Found 2 matches for "atmosphere.py" in /pvlib__pvlib-python/pvlib:
/pvlib__pvlib-python/pvlib/tests/atmosphere.py
/pvlib__pvlib-python/pvlib/atmosphere.py
```

Search Directory

```
search_dir APPARENT_ZENITH_MODELS
```

```
Found 4 matches for "APPARENT_ZENITH_MODELS" in /pvlib__pvlib-
python:
/pvlib__pvlib-python/docs/sphinx/source/whatsnew/v0.3.0.txt (1
matches)
/pvlib__pvlib-python/pvlib/atmosphere.py (2 matches)
/pvlib__pvlib-python/pvlib/location.py (1 matches)
End of matches for "APPARENT_ZENITH_MODELS" in /pvlib__pvlib-
python
```

Search File

```
search_file APPARENT_ZENITH_MODELS
```

```
Found 2 matches for "APPARENT_ZENITH_MODELS" in /pvlib__pvlib-
python/pvlib/atmosphere.py:
Line 12:APPARENT_ZENITH_MODELS = ('simple', 'kasten1966',
'kastenyoung1989',
Line 15:AIRMASS_MODELS = APPARENT_ZENITH_MODELS +
TRUE_ZENITH_MODELS
End of matches for "APPARENT_ZENITH_MODELS" in /pvlib__pvlib-
python/pvlib/atmosphere.py
```

2. File viewer

File Viewer

```
open atmosphere.py
```

```
(255 more lines above)
256:def gueymard94_pw(temp_air, relative_humidity):
257:  r"""
258:  Calculates precipitable water (cm) from ambient air temperature (C)
259:  and relatively humidity (%) using an empirical model. The
260:  accuracy of this method is approximately 20% for moderate PW (1-3
261:  cm) and less accurate otherwise.
262:
263:  The model was developed by expanding Eq. 1 in [2]_:
```

3. File editor

Response (Thought + Edit Action)

We need to add a check to ensure that all powers are the same. Let's edit the code to include this check.

```
edit 404:407
    elif (total_degree > 3 and
          len(set(k.exp[k.is_Pow])) == 1):
        if all(coeff[k] == 1):
            diop_type = 'general_sum_even'
end_of_edit
```

4. Context management

- Informative prompts
- Error messages
- History processors

* The paper discusses about interacting with a Linux Shell, but details are skipped in the presentation.

Experiment & Results: Main*

Table 1: Main results for SWE-agent performance on the full and Lite splits of the SWE-bench test set. We benchmark models in the SWE-agent, Basic CLI, and Retrieval Augmented Generation (RAG) settings established in SWE-bench [20].

	SWE-bench		SWE-bench Lite	
Model	% Resolved	$ Avg. Cost	% Resolved	$ Avg. Cost
RAG				
w/ GPT-4 Turbo	1.31	0.13	2.67	0.13
w/ Claude 3 Opus	3.79	0.25	4.33	0.25
Shell-only agent				
w/ GPT-4 Turbo	-	-	11.00	1.46
w/o Demonstration	-	-	7.33	0.79
SWE-agent				
w/ GPT-4 Turbo	**12.47**	1.59	**18.00**	1.67
w/ Claude 3 Opus	10.46	2.59	13.00	2.18

(paper put on arXiv: 2024-05-30)

Leaderboard

Lite | Verified | Full

Current results!

Model	% Resolved	Date
Honeycomb	22.06	2024-08-20
Amazon Q Developer Agent (v20240719-dev)	19.75	2024-07-21
Factory Code Droid	19.27	2024-06-17

Leaderboard

Lite | Verified | Full

Model	% Resolved	Date
CodeStory Aide + Mixed Models	43.00	2024-07-02
Honeycomb	38.33	2024-08-20
AbanteAI MentatBot + GPT 4o (2024-05-13)	38.00	2024-06-27

* The paper also provides Pass@1 results on HumanEvalFix and Pass@k details on SWE-bench Lite.

Experiment & Results: Ablation Study

Guardrails can improve error recovery

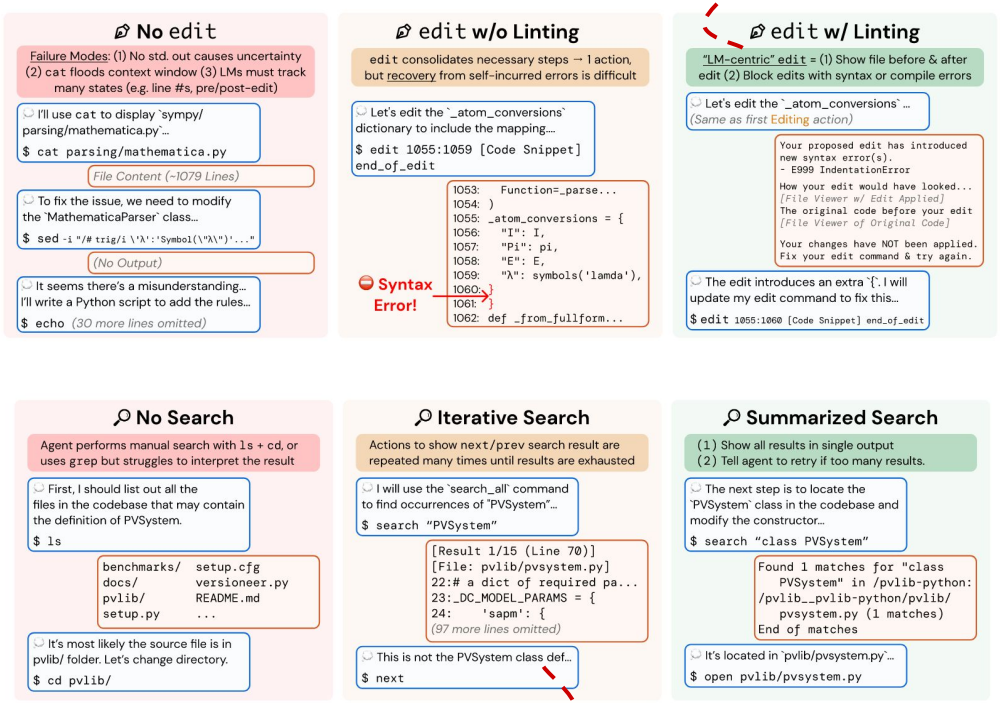
Editor	
`edit` action	15.0 ↓ 3.0
w/ linting	18.0
No `edit`	10.3 ↓ 7.7

Search	
Summarized	18.0
Iterative	12.0 ↓ 6.0
No search	15.7 ↓ 2.3

File Viewer	
30 lines	14.3 ↓ 3.7
100 lines	18.0
Full file	12.7 ↓ 5.3

Context	
Last 5 Obs.	18.0
Full history	15.0 ↓ 3.0
w/o demo.	16.3 ↓ 1.7

Compact and efficient design is better



Human user interfaces are not always suitable as agent-computer interfaces

Experiment & Results: Agent Behavior Analysis

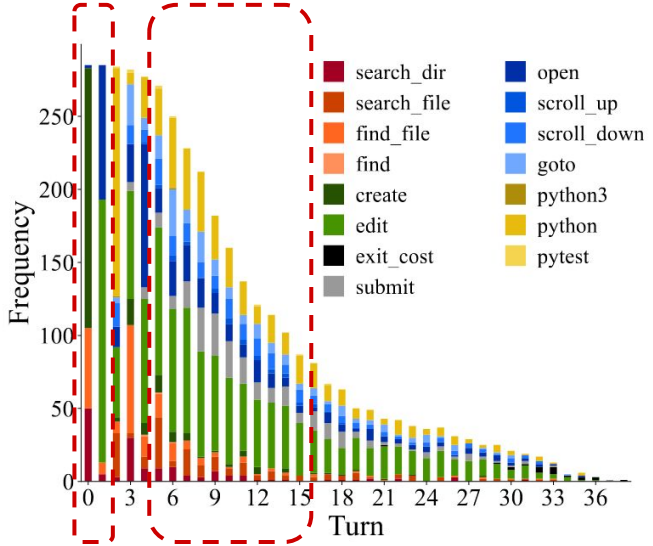


Figure 7: The frequency with which actions are invoked at each turn by SWE-agent w/ GPT-4 for task instances that it solved on the SWE-bench full test set (286 trajectories).

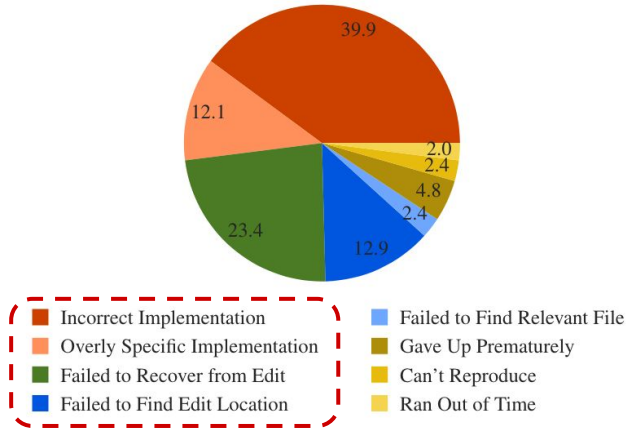


Figure 8: Failure mode distribution for SWE-agent w/ GPT-4 Turbo trajectories of unresolved instances. Each instance is labeled automatically using an LM with the categories from Table 9.

Thank you!
Q&A

Recap: This paper introduce SWE-agent, an agent composed of an LM and ACI capable of autonomously solving software engineering tasks!

Scientific Peer Reviewer

Sean McLeish

Summary

1. Edit
    a. Must edit whole lines
    b. Linting applied to check edits
2. Search
    a. Returns up to 50 results
    b. Asks agent again if >50 results
3. Viewer
    a. At most 100 lines shown
4. Context
    a. Only last 5 turns in context

- 51.7% of 2,294 problems have >1 linting error
- Average of 12 steps if successful vs 21 steps if unsuccessful
- Majority of failures are edit related

Table 3: SWE-bench Lite performance under ablations to the SWE-agent interface, which is denoted by 😈. We consider different approaches to searching and editing (see Figures 5 and 6, respectively). We also verify how varying the file viewer window size affects performance, and we ablate the effect of different context management approaches.

Editor		Search		File Viewer		Context	
`edit` action	15.0 ↓ 3.0	Summarized 😈	18.0	30 lines	14.3 ↓ 3.7	Last 5 Obs. 😈	18.0
w/ linting 😈	18.0	Iterative	12.0 ↓ 6.0	100 lines 😈	18.0	Full history	15.0 ↓ 3.0
No `edit`	10.3 ↓ 7.7	No search	15.7 ↓ 2.3	Full file	12.7 ↓ 5.3	w/o demo.	16.3 ↓ 1.7

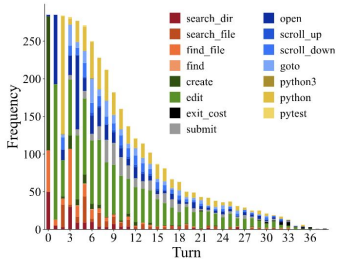


Figure 7: The frequency with which actions are invoked at each turn by SWE-agent w/ GPT-4 for task instances that it solved on the SWE-bench full test set (286 trajectories).

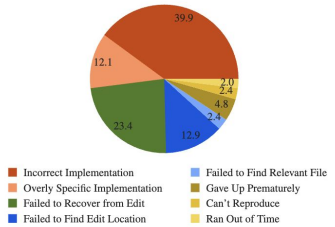


Figure 8: Failure mode distribution for SWE-agent w/ GPT-4 Turbo trajectories of unresolved instances. Each instance is labeled automatically using an LM with the categories from Table 9.

Strengths

- Increase in accuracy
  - 11.16% on SWE-Bench vs RAG
- Works across multiple programming languages
- Offers insight on LLM code generation as a whole with evaluation of failure cases
- Intuitive approach to the problem
- Doesn’t require training

Table 2: Pass@1 results on HumanEvalFix [32]. Except for SWE-agent, we use scores as reported in Yu et al. [65].

Model	Python	JS	Java
CodeLLaMa-instruct-13B	29.2	19.5	32.3
GPT-4	47.0	48.2	50.0
DeepseekCoder-CodeAlpaca-6.7B	49.4	51.8	45.1
WaveCoder-DS-6.7B	57.9	52.4	57.3
SWE-agent w/ GPT-4 Turbo	**87.7**	**89.7**	**87.9**

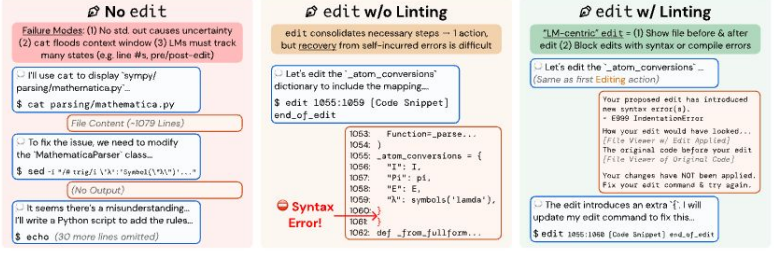
	SWE-bench		SWE-bench Lite	
Model	% Resolved	$ Avg. Cost	% Resolved	$ Avg. Cost
RAG				
w/ GPT-4 Turbo	1.31	0.13	2.67	0.13
w/ Claude 3 Opus	3.79	0.25	4.33	0.25
Shell-only agent				
w/ GPT-4 Turbo	-	-	11.00	1.46
w/o Demonstration	-	-	7.33	0.79
SWE-agent				
w/ GPT-4 Turbo	**12.47**	1.59	**18.00**	1.67
w/ Claude 3 Opus	10.46	2.59	13.00	2.18

Weaknesses

- Doesn’t work for current open source (‘weaker’) models
  - Authors “found their performance in the agent setting to be subpar”
- Is longer context open source models e.g. llama-128k
- More expensive than other methods (8-13x more costly)
  - More compute intensive
- Uses linting to reject incorrect edits
- >80% failure on SWE-Bench
- Could use fine-tuning to improve performance
- Robustness not evaluated
  - Is code generated secure?

Editor	
`edit` action	15.0 ↓ 3.0
w/ linting	18.0
No `edit`	10.3 ↓ 7.7

	SWE-bench		SWE-bench Lite	
Model	% Resolved	$ Avg. Cost	% Resolved	$ Avg. Cost
RAG				
w/ GPT-4 Turbo	1.31	0.13	2.67	0.13
w/ Claude 3 Opus	3.79	0.25	4.33	0.25
Shell-only agent				
w/ GPT-4 Turbo	-	-	11.00	1.46
w/o Demonstration	-	-	7.33	0.79
SWE-agent				
w/ GPT-4 Turbo	**12.47**	1.59	**18.00**	1.67
w/ Claude 3 Opus	10.46	2.59	13.00	2.18



Scores

Technical Correctness:

- [1] No Apparent Flaws

- Scientific Contribution:

  - [1] Independent Confirmation of Important Results with Limited Prior Research

  - [2] Creates a New Tool to Enable Future Science

- Presentation

  - [1] No Apparent Flaws

- Recommended Decision

  - [3] Accept with Meta Review

- Reviewer Confidence

  - [2] Highly Confident (May be more details in appendix)

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

Archaeologist

Nishit Anand

Previous papers which influenced the current paper:

1. SWE-Bench: Can Language Models Resolve Real-World GitHub Issues?
2. ReAct: Synergizing Reasoning and Acting in Language Models

Current Paper:

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

Subsequent paper which was influenced by the current paper:

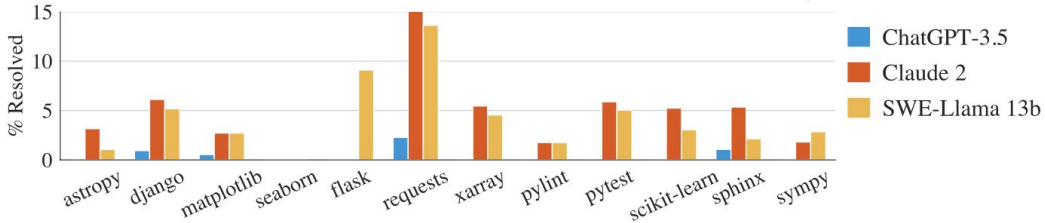
OpenDevin: An Open Platform for AI Software Developers as Generalist Agents

Previous Papers

1. **SWE-Bench: Can Language Models Resolve Real-World GitHub Issues?**

Summary of the paper:

- The paper introduces SWE-Bench, a new benchmark dataset comprising 2294 software engineering problems from 12 popular python repositories, with the aim of evaluating LLM models on real world software engineering tasks
- Claude 2 performed the best, solving only 1.96% of the Github issues in the benchmark
- The authors own model SWE-Llama, which is actually CodeLlama fine-tuned on a separate dataset of 19000 Github issues-PR pairs, achieved same level of performance as Claude 2 in ‘oracle’ setting i.e., when the model knows which files were actually edited



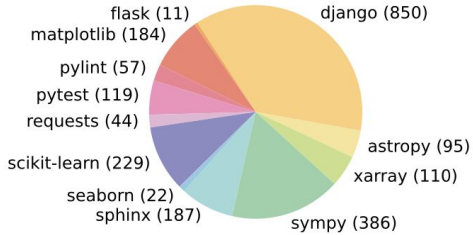


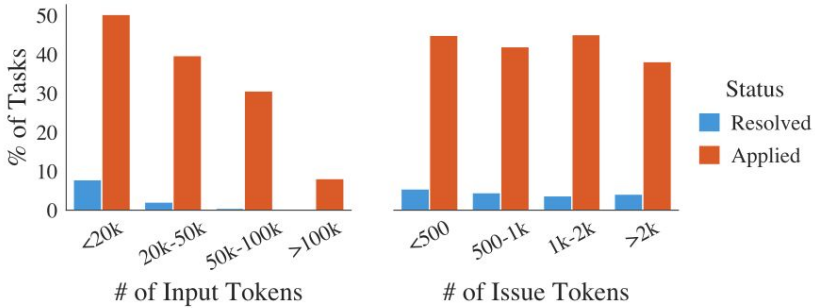
Figure 3: Distribution of SWE-bench tasks (in parenthesis) across 12 open source GitHub repositories that each contains the source code for a popular, widely downloaded PyPI package.

- The findings of the paper were that model performance decreases with increase in context length and that the models struggle with localizing problematic code in larger codebases
- According to the authors, SWE-bench provides a realistic and challenging environment for evaluation and improvement of LLMs in the context of software engineering tasks and that advances in SWE-Bench would represent steps towards LLMs that are ore practical, autonomous and intelligent

Leaderboard

Lite | Verified | Full

Model	% Resolved	Date
Honeycomb	22.06	2024-08-20
Amazon Q Developer Agent (v20240719-dev)	19.75	2024-07-21
Factory Code Droid	19.27	2024-06-17
AutoCodeRover (v20240620) + GPT 4o (2024-05-13)	18.83	2024-06-28
✅ SWE-agent + Claude 3.5 Sonnet	18.13	2024-06-20
✅ AppMap Navie + GPT 4o (2024-05-13)	14.60	2024-06-15
Amazon Q Developer Agent (v20240430-dev)	13.82	2024-05-09



How this paper is related to the current paper?

- The current paper: SWE-Agent, uses the SWE-Bench benchmark for evaluation of their proposed method: SWE-Agent, which is their ACI and LLM combination
- They also use the SWE-Bench benchmark to test and compare their method with a RAG-based approach and a Shell-Only approach for solving software engineering tasks
- SWE-Bench is the main benchmark (apart from the HumanEval Fix benchmark) on which the SWE-Agent paper is based and the SWE-Agent paper would not have been possible without the SWE-Bench paper, thus the SWE-Bench paper plays a crucial role in the SWE-Agent paper
- Also, both the papers are written by the same authors and are from the NLP group led by Prof. Karthik Narasimhan at Princeton

Table 1: Main results for SWE-agent performance on the full and Lite splits of the SWE-bench test set. We benchmark models in the SWE-agent, Basic CLI, and Retrieval Augmented Generation (RAG) settings established in SWE-bench [20].

	SWE-bench		SWE-bench Lite	
Model	% Resolved	$ Avg. Cost	% Resolved	$ Avg. Cost
RAG				
w/ GPT-4 Turbo	1.31	0.13	2.67	0.13
w/ Claude 3 Opus	3.79	0.25	4.33	0.25
Shell-only agent				
w/ GPT-4 Turbo	-	-	11.00	1.46
w/o Demonstration	-	-	7.33	0.79
SWE-agent				
w/ GPT-4 Turbo	**12.47**	1.59	**18.00**	1.67
w/ Claude 3 Opus	10.46	2.59	13.00	2.18

Previous Paper 2 - ReAct: Synergizing Reasoning and Acting in Language Models

Summary:

- The authors introduce ReAct, an approach that combines reasoning and acting capabilities in LLMs.
- Existing methods typically focus on either reasoning (e.g. chain-of-thought prompting) or acting (e.g. action generation for interactive environments) separately
- ReAct aims to synergize these two capabilities to improve performance and interpretability

Approach:

- Prompts LLMs to generate both verbal reasoning traces and task-specific actions in an interleaved manner
- Reasoning helps create and adjust plans for acting, while actions allow interaction with external environments to gather information

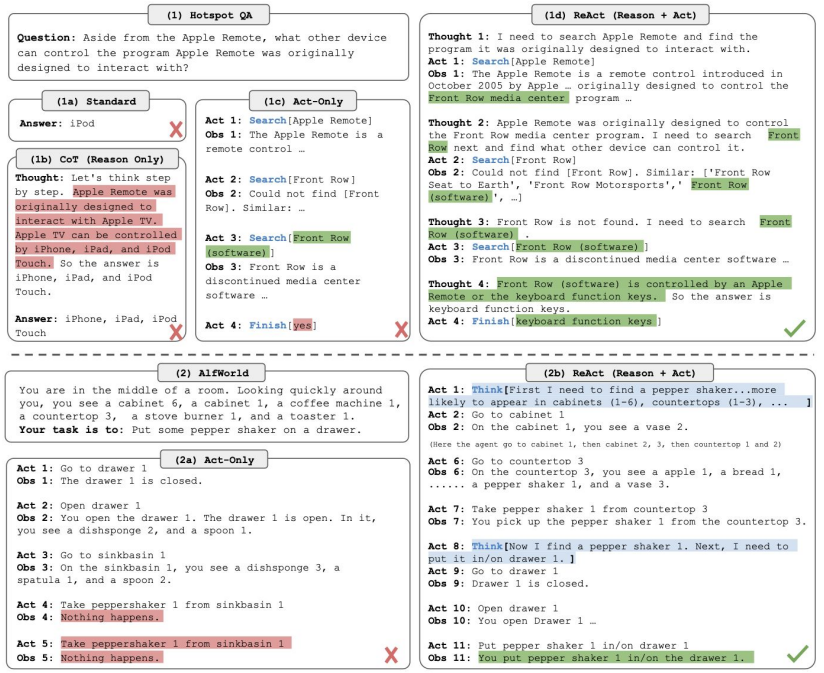


Figure 1: (1) Comparison of 4 prompting methods, (a) `Standard`, (b) Chain-of-thought (`CoT`, Reason Only), (c) `Act`-only, and (d) `ReAct` (Reason+Act), solving a HotpotQA (Yang et al., 2018) question. (2) Comparison of (a) `Act`-only and (b) `ReAct` prompting to solve an AlfWorld (Shridhar et al., 2020b) game. In both domains, we omit in-context examples in the prompt, and only show task solving trajectories generated by the model (Act, Thought) and the environment (Obs).

Findings:

1. ReAct outperforms action-only baselines consistently across all tasks
2. Combining ReAct with CoT achieves the best performance for prompting LLMs.
3. Improves action planning and goal tracking in decision-making tasks through reasoning.

How this paper is related to the current paper?

- SWE-Agent paper is influenced by and uses ReAct in their paper
- The idea of Thought and Action in every instruction given by the LLM is taken from the ReAct paper

Observation (File Viewer)

```
[File: solvers/diophantine.py (3093 lines total)]
(400 lines above)
401:    elif total_degree == 3:
402:        diop_type = "cubic_thue"
403:
404:    elif (total_degree > 3):
405:        if all(coeff[k] == 1):
406:            diop_type = 'general_sum_even'
407:
408:    if diop_type is not None:
409:        return var, coeff, diop_type
(2684 lines below)
```

(a) Observation from the file viewer.

Response (Thought + Edit Action)

We need to add a check to ensure that all powers are the same. Let's edit the code to include this check.

```
edit 404:407
    elif (total_degree > 3 and
          len(set(k.exp[k.is_Pow])) == 1):
        if all(coeff[k] == 1):
            diop_type = 'general_sum_even'
end_of_edit
```

(b) Action using the edit interface.

SWE-agent **pylint-dev__pylint-5859**

It appears that the issue persists because the search term is still being interpreted as an option. To circumvent this, we can use a different search term that is less likely to be confused with a command-line option. Let's try searching for just "notes" instead, which should still lead us to the relevant part of the code.

```
search_dir "notes"
```

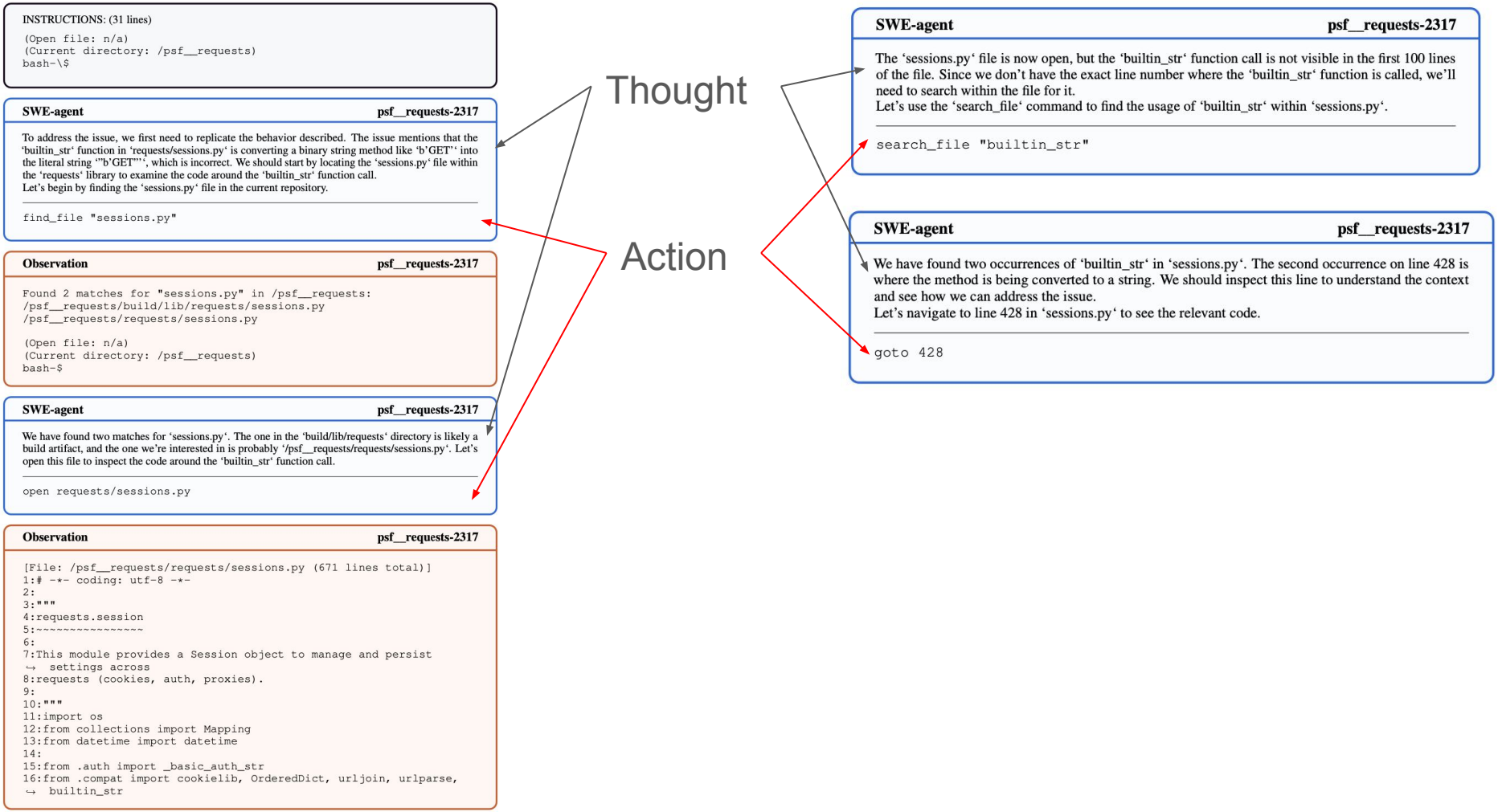
Thought

Action

SWE-agent **psf__requests-2317**

We are now looking at the section of the 'sessions.py' file where the 'builtin_str' function is used to convert the 'method' variable. According to the issue, this conversion is causing a problem when the 'method' is a binary string because it's being converted to a literal string that includes the binary prefix 'b'...''.
To address this issue, we need to ensure that the 'method' variable is correctly converted to a string without the binary prefix. This might involve checking the type of the 'method' variable and handling binary strings differently from regular strings.
Let's edit the line to add a check that ensures binary strings are decoded to regular strings before being passed to 'builtin_str'.

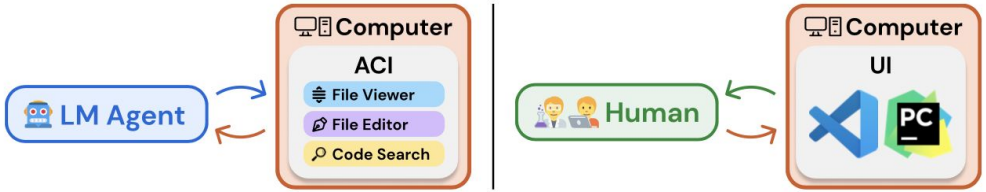
```
edit 428:428
        # Ensure method is a string, decode if it's a binary string
        if isinstance(method, bytes):
            method = method.decode('ascii')
        method = builtin_str(method)
end_of_edit
```

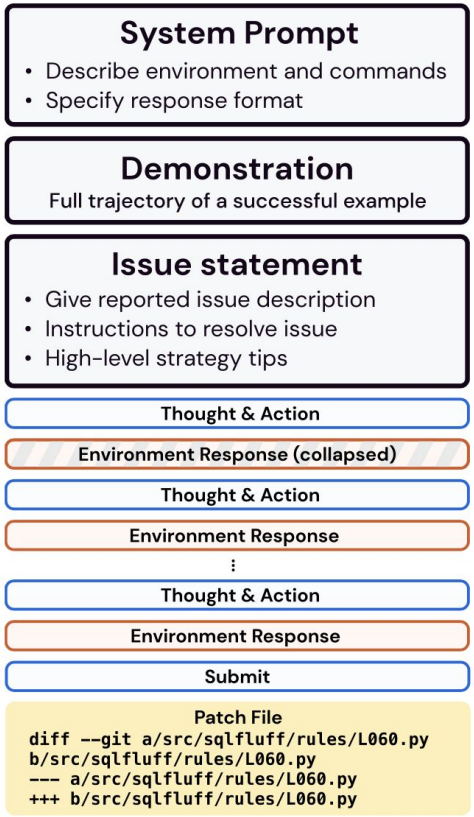

INSTRUCTIONS: (31 lines)
(Open file: n/a)
(Current directory: /psf__requests)
bash-\$
SWE-agent psf__requests-2317
To address the issue, we first need to replicate the behavior described. The issue mentions that the 'builtin_str' function in 'requests/sessions.py' is converting a binary string method like 'b'GET'' into the literal string '"b'GET"'', which is incorrect. We should start by locating the 'sessions.py' file within the 'requests' library to examine the code around the 'builtin_str' function call.
Let's begin by finding the 'sessions.py' file in the current repository.
find_file "sessions.py"
Observation psf__requests-2317
Found 2 matches for "sessions.py" in /psf__requests:
/psf__requests/build/lib/requests/sessions.py
/psf__requests/requests/sessions.py
(Open file: n/a)
(Current directory: /psf__requests)
bash-$
SWE-agent psf__requests-2317
We have found two matches for 'sessions.py'. The one in the 'build/lib/requests' directory is likely a build artifact, and the one we're interested in is probably '/psf__requests/requests/sessions.py'. Let's open this file to inspect the code around the 'builtin_str' function call.
open requests/sessions.py
Observation psf__requests-2317
[File: /psf__requests/requests/sessions.py (671 lines total)]
1:# -*- coding: utf-8 -*-
2:
3:"""
4:requests.session
5:~~~~~~~~~~~~~~~~
6:
7:This module provides a Session object to manage and persist
↪ settings across
8:requests (cookies, auth, proxies).
9:
10:"""
11:import os
12:from collections import Mapping
13:from datetime import datetime
14:
15:from .auth import _basic_auth_str
16:from .compat import cookielib, OrderedDict, urljoin, urlparse,
↪ builtin_str
Thought
Action
SWE-agent psf__requests-2317
The 'sessions.py' file is now open, but the 'builtin_str' function call is not visible in the first 100 lines of the file. Since we don't have the exact line number where the 'builtin_str' function is called, we'll need to search within the file for it.
Let's use the 'search_file' command to find the usage of 'builtin_str' within 'sessions.py'.
search_file "builtin_str"
SWE-agent psf__requests-2317
We have found two occurrences of 'builtin_str' in 'sessions.py'. The second occurrence on line 428 is where the method is being converted to a string. We should inspect this line to understand the context and see how we can address the issue.
Let's navigate to line 428 in 'sessions.py' to see the relevant code.
goto 428

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

Summary:

- According to the authors, existing user interfaces for code have been designed with humans as end users in mind and that LLM agents represent a different end user, and could benefit from better-designed interfaces for performing software engineering tasks
- They feel ACI (Agent-Computer Interface) design aims to create a more suitable interface which would make LLM agents more effective at software engineering tasks

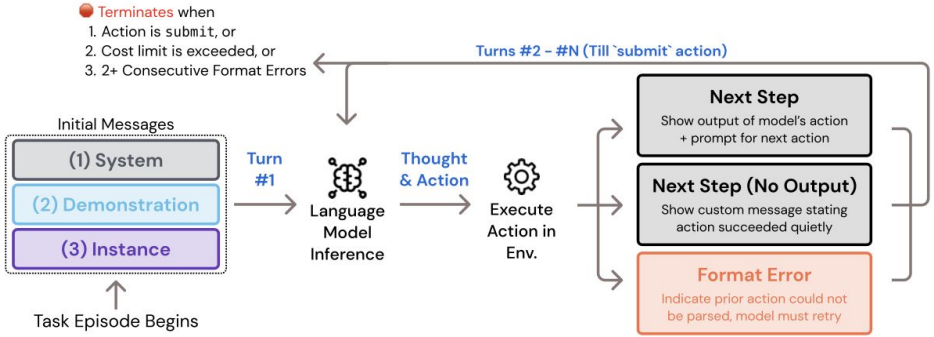




- To this end, the paper introduces SWE-Agent, which is a combination of ACI (Agent-Computer Interface) and a LLM that can interact with a computer to solve challenging real-world software engineering problems
- Their ACI comprises of several functionalities like Search and navigation, File Viewer, File Editor and Context Management, where the LLM generates both a thought and an action at each step
- Using GPT-4 Turbo as a base LM, SWE-agent achieves pass@1 score of 12.47% on the 2,294 SWE-bench test tasks, and 87.7% on HumanEvalFix which are both SOTA scores

Table 2: Pass@1 results on HumanEvalFix [32]. Except for SWE-agent, we use scores as reported in Yu et al. [65].

Model	Python	JS	Java
CodeLLaMa-instruct-13B	29.2	19.5	32.3
GPT-4	47.0	48.2	50.0
DeepseekCoder-CodeAlpaca-6.7B	49.4	51.8	45.1
WaveCoder-DS-6.7B	57.9	52.4	57.3
SWE-agent w/ GPT-4 Turbo	**87.7**	**89.7**	**87.9**



Editor		Search		File Viewer		Context	
`edit` action	15.0 ↓3.0	Summarized 🤠	18.0	30 lines	14.3 ↓3.7	Last 5 Obs. 🤠	18.0
w/ linting 🤠	18.0	Iterative	12.0 ↓6.0	100 lines 🤠	18.0	Full history	15.0 ↓3.0
No `edit`	10.3 ↓7.7	No search	15.7 ↓2.3	Full file	12.7 ↓5.3	w/o demo.	16.3 ↓1.7

Subsequent paper which was influenced by the current paper

OpenDevin: An Open Platform for AI Software Developers as Generalist Agents

Summary:

- In the paper, authors introduce OpenDevin which is a community-driven platform designed for the development of generalist and specialist AI agents that interact with the world through software

The State and Event Stream:

- In OpenDevin, the state is a data structure that encapsulates all relevant information for the agent's execution
- The event stream is a chronological collection of past actions and observations, including the agent's own actions and user interactions (e.g. instructions and feedback)
- Observations: Observations describe environmental changes that the agent observes. It can be either messages from the user instructing agents to perform certain tasks or the execution outcome of the agent's previous action
- Implementing a New Agent: The agent abstraction allows users to create and customize agents for various tasks easily. The core of the agent abstraction lies in the **step** function, which takes the current state as input and generates an appropriate action based on the agent's logic

Figure 2: Minimal example of implementing an agent in OpenDevin.

```python
class MinimalAgent:
    def reset(self) -> None:
        self.system_message = "You are a helpful assistant ..."

    def step(self, state: State):
        messages: list[dict[str, str]] = [
            {'role': 'system', 'content': self.system_message}
        ]
        for prev_action, obs in state.history:
            action_message = get_action_message(prev_action)
            messages.append(action_message)
            obs_message = get_observation_message(obs)
            messages.append(obs_message)

        # use llm to generate response (e.g., thought, action)
        response = self.llm.do_completion(messages)

        # parse and execute action in the runtime
        action = self.parse_response(response)
        if self.is_finish_command(action):
            return AgentFinishAction()
        elif self.is_bash_command(action):
            return CmdRunAction(command=action.command)
        elif self.is_python_code(action):
            return IPythonRunCellAction(code=action.code)
        elif self.is_browser_action(action):
            return BrowseInteractiveAction(code=action.code)
        else:
            return MessageAction(content=action.message)
```

Browser Access and **Actions**:

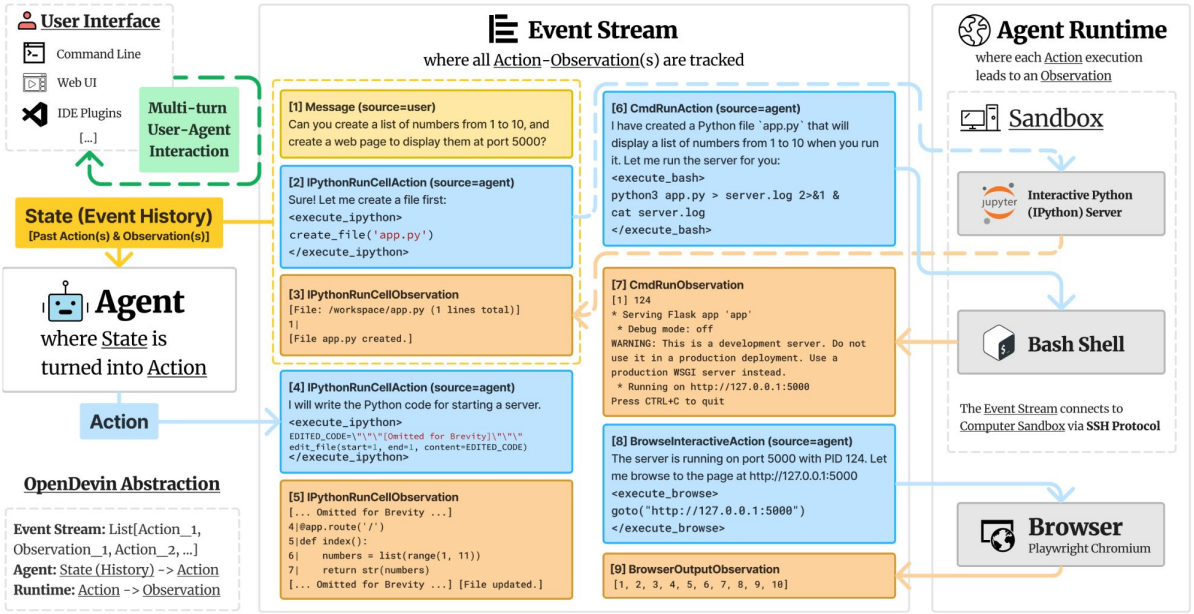
- The authors introduce an environment which consists of a sandboxed OS and browser which agents can use for their tasks
- **IPythonRunCellAction** and **CmdRunAction** enable the agent to execute arbitrary Python code and bash commands inside the sandbox environment (e.g. a securely isolated Linux operating system)
- **BrowserInteractiveAction** enables interaction with a web browser with a domain-specific language
- **Improvement over SWE-Agent which doesn't have browser access**

Agent Runtime:

- Linux SSH Sandbox - OpenDevin connects to the sandbox through SSH protocol, executes arbitrary commands from the agent, and returns the execution results as observations to the agent.
- Jupyter IPython
- Web Browser using BrowserGym

AgentSkills: The Extensible Agent-Computer Interface

- AgentSkills is designed as a Python package consisting of different utility functions that are automatically imported into the Jupyter IPython environment. The ease of defining a Python function as a tool lowers the barrier for community members to contribute new tools to the skill library



- **AgentSkills library includes file editing utilities adapted from SWE-Agent like edit_file**, which allows modifying an existing file from a specified line; scrolling functions **scroll_up** and **scroll_down** for viewing a different part of files. Also, parse_image and parse_pdf for GPT-4V

Agent Delegation: Cooperative Multi-agent Interaction -

- Another unique feature is that there is multi-agent delegation, which allows multiple specialized agents to work together
- Special action type **AgentDelegateAction**, which enables an agent to delegate a specific subtask to another agent
- CodeActAgent can delegate tasks like navigate the web, click buttons, submit forms to BrowsingAgent
- **Micro Agent**: A micro agent re-uses most implementations from an existing generalist agent (e.g., CodeAct Agent). It is designed to lower the barrier to agent development, where community members can share specialized prompts that work well for their particular use cases and can create a Micro Agent without programming

Evaluation:

- They also use an evaluation framework, facilitating the evaluation of agents across a wide range of tasks

Table 2: Evaluation benchmarks in OpenDevin.

Category	Benchmark	Required Capability
Software	SWE-Bench [21]	Fixing Github issues
	HumanEvalFix [37]	Fixing Bugs
	BIRD [27]	Text-to-SQL
	BioCoder [58]	Bioinformatics coding
	ML-Bench [57]	Machine learning coding
	Gorilla APIBench [46]	Software API calling
	ToolQA [81]	Tool use
Web	WebArena [79]	Goal planning & realistic browsing
	MiniWoB++ [30]	Short trajectory on synthetic web
Misc. Assistance	GAIA [34]	Tool-use, browsing, multi-modality
	GPQA [50]	Graduate-level Google-proof Q&A
	AgentBench [31]	Operating system interaction (bash)
	MINT [64]	Multi-turn math and code problems
	Entity Deduction Arena [77]	State tracking & strategic planning
	ProofWriter [55]	Deductive Logic Reasoning

How this paper is related to the current paper?

- This paper builds on the work of SWE-Agent paper. It builds a bigger and better LLM agent interaction framework in which multiple agents can work together and also allows an OS and browser access, so that the agents can access the web. This paper cites the SWE-Agent paper as well.
- The authors compare their method's performance to SWE-Agent in most of the tables and many of the ideas in the paper are influenced by the SWE-Agent paper and are an improved version of what was proposed in the SWE-Agent Paper
- OpenDevin directly borrows some tasks/actions from SWE-Agent like edit_file, scroll_up, scroll_down
- It improves the functionality of LLM agents and allows LLMs to push their performance on SWE-Bench and other benchmarks. This would not have been possible without the SWE-Bench and SWE-Agent paper in the first place

Framework	Domain	Graphic User Interface	Standardized Tool Library	Built-in Sandbox & Code Execution	Built-in Web Browser	Multi-agent Collaboration	Human-AI Collaboration	AgentHub	Evaluation Framework	Agent QC
AutoGPT [14]	General	✔	✗	✗	✗	✗	✗	✔	✗	✔
LangChains [6]	General	✗	✔	✗*	✗*	✗	✗	✔	✗	✗
MetaGPT [16]	General	✗	✔	✗	✔	✔	✗	✔	✗	✔
AutoGen [67]	General	✗	✔	✔	✔	✔	✔	✔	✔	✗
AutoAgents [7]	General	✗	✗	✗	✗	✔	✗	✗	✗	✗
Agents [80]	General	✗	✗	✗	✗	✔	✔	✗	✗	✗
Xagents [61]	General	✔	✔	✗	✔	✔	✗	✔	✗	✗
OpenAgents [69]	General	✔	✗	✔	✔	✗	✗	✔	✗	✗
GPTSwarm [83]	General	✗	✔	✗	✗	✔	✔	✗	✗	✗
AutoCodeRover [78]	SWE	✗	✗	✔	✗	✗	✗	✗	✗	✗
SWE-Agent [72]	SWE	✗	✗	✔	✗	✗	✗	✗	✗	✗
OpenDevin	General	✔	✔	✔	✔	✔	✔	✔	✔	✔

Agent	Model	Software (§4.2) SWE-Bench Lite	Web (§4.3) WebArena	Misc. (§4.4) GPQA	GAIA
		Software Engineering Agents			
SWE-Agent [72]	`gpt-4-1106-preview`	18.0	–	–	–
AutoCodeRover [78]	`gpt-4-0125-preview`	19.0	–	–	–
Aider [13]	`gpt-4o` & `claude-3-opus`	26.3	–	–	–
Moatless Tools [85]	`claude-3.5-sonnet`	26.7	–	–	–
Agentless [68]	`gpt-4o`	27.3	–	–	–
		Web Browsing Agents			
Lemur [70]	`Lemur-chat-70b`	–	5.3	–	–
Patel et al. [45]	Trained 72B w/ synthetic data	–	9.4	–	–
AutoWebGLM [24]	Trained 7B w/ human/agent annotation	–	18.2	–	–
Auto Eval & Refine [42]	GPT-4 + Reflexion w/ GPT-4V	–	20.2	–	–
WebArena Agent [79]	`gpt-4-turbo`	–	14.4	–	–
		Misc. Assistance Agents			
AutoGPT [14]	`gpt-4-turbo`	–	–	–	13.2
Few-shot Prompting + Chain-of-Thought [50]	`Llama-2-70b-chat`	–	–	28.1	–
	`gpt-3.5-turbo-16k`	–	–	29.6	–
	`gpt-4`		–	38.8	–
		OpenDevin Agents			
CodeActAgent `v1.8`	`gpt-4o-mini-2024-07-18`	6.3	8.3	–	–
	`gpt-4o-2024-05-13`	22.0	14.5	*53.1	–
	`claude-3-5-sonnet`	26.0	15.3	52.0	–
GPTSwarm `v1.0`	`gpt-4o-2024-05-13`	–	–	–	32.1

Academic Researcher

Raman

Example of an issue in codebase

notifications.py

```
def get_user_details(user_id):
    return info_from_db(user_id)

def send_alert(user_id):
    user_details = get_user_details(user_id)
    alert(user_details['email'])
```

Issue: As database grows, it becomes more expensive to get user details

Potential Solution

notifications.py

```
def get_user_details(user_id):
    if user_id not in user_cache:
        user_cache[user_id] = info_from_db(user_id)
    return user_cache[user_id]

def send_alert(user_id):
    user_details = get_user_details(user_id)
    alert(user_details['email'])
```

Add a cache that stores user details to avoid an expensive DB call

Do you spot an issue in in this module?

authentication.py

```
def authenticate(user_id, given_password):
    user_details = get_user_details(user_id)
    return check_password(args)
```

LLMs are quite good at generating code

T	Model	Win Rate	humaneval-python
EXT	Nxcode-CQ-7B-orpo	55.42	87.23
	CodeQwen1.5-7B-Chat	55.08	87.2
EXT	DeepSeek-Coder-7b-instruct	50.33	80.22
EXT	DeepSeek-Coder-33b-instruct	52	80.02
EXT	CodeFuse-DeepSeek-33b	54.33	76.83
	CodeLlama-70b-Instruct	43.58	75.6
EXT	OpenCodeInterpreter-DS-33B	55.83	75.23
EXT	OpenCodeInterpreter-DS-6.7B	49.67	73.2

Repository level coding is challenging

🤠✅ SWE-agent + Claude 3.5 Sonnet	18.13
🤠✅ AppMap Navie + GPT 4o (2024-05-13)	14.60
Amazon Q Developer Agent (v20240430-dev)	13.82
🤠✅ SWE-agent + GPT 4 (1106)	12.47
🤠✅ SWE-agent + GPT 4o (2024-05-13)	11.99
🤠✅ SWE-agent + Claude 3 Opus	10.51
🤠✅ RAG + Claude 3 Opus	3.79
🤠✅ RAG + Claude 2	1.96
🤠✅ RAG + GPT 4 (1106)	1.31
🤠✅ RAG + SWE-Llama 13B	0.70
🤠✅ RAG + SWE-Llama 7B	0.70
🤠✅ RAG + ChatGPT 3.5	0.17

Do LLMs guarantee secure code generation?

Leaderboard

Rank	Model	pass@1	secure@1$_{pass}$	secure-pass@1
1	GPT-4-1106-preview	70.13	57.97	47.45
2	DeepseekCoder-33B	78.77	56.09	46.54
3	Llama3-8B	74.37	57.88	46.54
4	CodeLlama-34B	75.47	53.51	44.53
5	SafeCoder-Mistral-7B-v0.1	63.26	62.08	44.43
6	CodeGemma-7B	73.93	54.34	43.64
7	Mistral-7B-v0.1	73.32	54.41	41.15
8	CodeLlama-7B	67.13	55.3	39.76

Can we ensure SWE Agent generates secure code?

Broad Idea: Improve security aspect of SWE Agents at a repository level.

Concretely: The follow-up seeks to evaluate and improve the ability of LLM in generating secure and functionally correct code at repository level, where a key challenge is inter-procedural data and control flow.

Challenges: What do we measure on and how to incorporate security checks

What do we measure on? Repo level security benchmark

1) VulEval is a recently developed benchmark that focuses on inter-procedural vulnerability detection.
2) It is not yet a benchmark which has issues that can be solved by LLM agents.
3) There are two ways to modify this benchmark
    a) Use the commit that introduced vulnerability as a issue to be solved
    b) Introduce a new issue that has to use part of vulnerable code

VulEval: Towards Repository-Level Evaluation of Software Vulnerability Detection

How to improve agent's security capability?

1) LLMs by themselves are not quite good at patching security issues - use feedback from static analyzers such as Bandit
2) Tools like dependabot can also be used to track vulnerabilities in dependencies or can be connected to an updated source of new vulnerabilities (RAG setup)
3) A new action in SWE Agent after proposing solution should be to track flow of control and data, to check if there are any unintended vulnerabilities introduced

Industry Practitioner

Srividya Ponnada

SWE-Agent for Industrial Adoption

Advantages:

- **Automation of Repetitive Tasks**, such as boilerplate code generation, CRUD operations, and refactoring, freeing developers for complex tasks
- **Improves Efficiency**, speeds up the SDLC; automates bug-fixing and refactoring, improving team productivity
- **Reduced Errors & Consistency**, minimizes human error and ensures adherence to coding standards across the codebase
- **Long-Term Cost Efficiency**, high initial investment but lowers labor costs and reduces delays over time
- **Competitive Advantage**, faster delivery of new features, enhancing market agility

Challenges:

- **High Initial & Operational Costs**, significant computational resource requirements
- **Dependency on LLMs,** performance relies on model quality and adaptability to specific domains
- **Complexity & Learning Curve**, requires a team with AI/ML expertise
- **Ethical & Security Concerns,** concerns over bias, job displacement, and risks in sensitive projects

Should We Adopt SWE-Agent?

When to Adopt:

- **Large-scale enterprises** with complex, repetitive software development tasks
- **Teams seeking to reduce human error** and maintain coding consistency
- **Organizations planning for scalability** and long-term ROI from automation

When Not to Adopt:

- **Small-to-medium-sized teams** with limited budgets and computational resources
- **Projects needing highly specialized knowledge** or facing ethical concerns (AI bias, job displacement)
- **Teams without sufficient AI/ML expertise** or those unwilling to invest in necessary modifications

Conclusion:

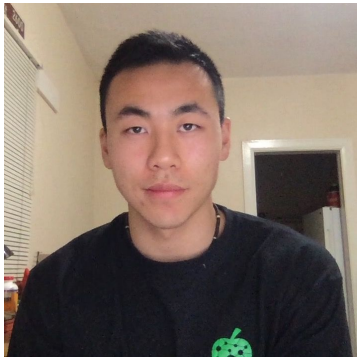
Organizations must weigh the benefits against the challenges, and ensure they have the infrastructure, expertise, and ethical frameworks in place to maximize its advantages. By adopting a structured and well-supported approach, organizations can leverage SWE-agent to accelerate software development, enhance productivity, and maintain a competitive position. However, for smaller teams or projects with unique requirements, may look into alternative solutions.

Private Investigator

Arthur Drake and Manan Suri

9 September 2024

First Author: John Yang | Stanford University



- Researcher at **Meta**, entering 1st year as a CS PhD at **Stanford University.**
- Completed CS MS at **Princeton University** where he developed SWE-agent.
    - Advisor: Karthik Narasimhan
- Completed CS BS at **UC Berkeley.**

Primary Research Areas: Language Agents; Language Model Evaluation; SWE.

Previous Works: Helped Develop **SWE-Bench** in 2023: New dataset with ~2,300 real-world SWE problems drawn from real GitHub issues. Created **InterCode** in 2023 which is a new LLM-based framework for interactive coding using Reinforcement Learning.

Motivation: John has a long history of interest in ML and NLP, dating back to 2017 as seen from his blog at john-b-yang.github.io. He has an advanced knowledge of the current literature and has even posted public notes on many papers he's read. Finally, he has taught several CS/ML courses at Princeton and UC Berkeley.

Sixth Author: Karthik Narasimhan | Princeton

- Associate Professor at **Princeton** and Head of Research at **Sierra.**
- Completed MS CS and PhD in CS from **MIT**.
- Completed B.Tech. in CSE from **Indian Institute of Technology, Madras**.
- Previously, spent an year as a Research Scientist at OpenAI, during which time he built the original GPT.

Improving Language Understanding by Generative Pre-Training (GPT)

Authors	Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever
Publication date	2018
Source	https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf
Publisher	Technical report, OpenAI
Total citations	Cited by 11407

Sixth Author: Karthik Narasimhan | Princeton

Primary Research Areas: Language Agents, NLP, RL

Previous Works: Broadly, his work is in:

- **Language Models:** GPT
- **Language Agents:** some recent works include– SWE-Agent, CoALA (Cognitive Architectures for Language Agents), Reflexion: Language Agents with Verbal Reinforcement Learning
- **Benchmarks:** some recent works include– SWE-Bench, Intercode, WebShop, SILG, TAU Bench
- **Efficiency and Safety:** DataMUX, Toxicity in ChatGPT
- **Reinforcement Learning:** Multi-Objective RL, POLCO, XTX

Motivation: His research in language models and language agents speaks directly for a strong motivation for research in the direction of this paper. In this talk at the Open AGI Summit, he talks passionately about how current progresses in language models and decision making create an opportunity for autonomous agents that work with a human-in-the-loop in a collaborative framework, calling it a “dream for AI” and something that would “propel humankind forward”. Code, being a powerful tool that actualizes abstract concepts in a structured and efficient manner, is a natural avenue to build Language Agents on.

Role: Hacker

Henry Blanchette

as we saw earlier …

Agent-Computer Interface (ACI): Design Principles

(… draw from HCI)

1. Actions should be **simple and easy** to understand for agents
2. Actions should be **compact and efficient**
3. **Environment feedback** should be informative but concise
4. Guardrails **mitigate error** propagation and hasten recovery

… what if we added more kinds of actions?

More Actions

- The implementation provides only very **simple**, **compact**, **error-localizing** commands to the agent
- Basically, the agent has a little state machine representation of the environment, and has to do all the actual programming via it's own inherent ability
- But what if we interpret there restrictions a little more loosely? Then what's possible?
- **Research question:** How can we augment the agent's abilities with tools?

Experiment Setup

- Brainstorm some functionalities the agent doesn’t already have, and add them as commands in the ACI
- Run the augmented agent on simple examples to see if it works at all

- Related work: only the reference paper

Case Study: Translation

The functions in `src/utilities.py` are all named in Ipsentiloese, a rare language known by the original developers. To assist with readability, these names must be translated to English. You should use your `translate_ipsentiloese` command to get the English translations of each Ipsentiloese function name.

```
# @yaml
# signature: translate_ipsentiloese <file_path>
# docstring: Translate a string from Ipsentiloese to English.
# arguments:
#   input:
#       type: string
#       description: The string to translate
#       required: true
```

utilities.py src

```
-def rtl_ne_khrs(xs):
1+def sum_of_list(xs):
2     r = 0
3     for x in xs:
4         r += x
5     return r
6
7
-def oqnctbs_ne_khrs(xs):
8+def product_of_list(xs):
9     r = 1
10    for x in xs:
11        r *= x
12    return r
```

Case Study: Following Code Quality Conventions

Use the `auto_qa` command to check for any code quality warnings according to Ayeye Corp's in-house conventions. For each function the module `src/utilities.py`, use `auto_qa`. For each warning, fix the code to not have that warning anymore.

```
# @yaml
# signature: auto_qa <snippet>
# docstring: Does a code quality analysis, according to Ayeye Corp's in-house
#            conventions, on the given code snippet.
# arguments:
#   snippet:
#       type: string
#       description: The code snippet to be analyzed.
#       required: true
```

utilities.py src

```
-def a(x, y):
1+def add_numbers(x, y):
2     return x + y
3
4
-def c(bit1, bit2):
5+def toggle_bit(bit1, bit2):
6     if bit1:
7         return bit2
8     else:
9         return not bit2
10
11
-def f(predicate, list):
12+def filter_list(predicate, list):
13    new_list = []
14    for x in list:
15        if predicate(x):
16            new_list.append(x)
17    return new_list
18
```

Takeaways

- The agent seems to **understand human descriptions** of tools
- The agent usually needed **very obvious tells** for where to actually use the tools I wanted it to rather than its other tools
- The agent can **make sense of human-like outputs** from tools
- This could be a neat way to **“offload” tasks that are best done by algorithm** from the agent’s core logic loop, while still giving it access via a human-described ACI

Future Work

- **Quantify agent performance** when using other kinds of tools for particular tasks
  - Accuracy
  - Error recovery
- Investigate what kinds of tools and ACIs to the same tools are **best made use of** by the agent

Social Impact Assessor

Ethan Baker

Highlighted Positive Impacts

Improvements over traditional LMs:

- **Performance:** SWE-Agent outperforms static LMs in SWE-Bench, and could be applied to many of the same scenarios.

Table 2: Pass@1 results on HumanEvalFix [32]. Except for SWE-agent, we use scores as reported in Yu et al. [65].

Model	Python	JS	Java
CodeLLaMa-instruct-13B	29.2	19.5	32.3
GPT-4	47.0	48.2	50.0
DeepseekCoder-CodeAlpaca-6.7B	49.4	51.8	45.1
WaveCoder-DS-6.7B	57.9	52.4	57.3
SWE-agent w/ GPT-4 Turbo	**87.7**	**89.7**	**87.9**

Potential for Innovation:

- **New Applications:** Opportunities for applying ACIs and SWE-Agent techniques to new and diverse areas.

Positive Impacts Not Directly Mentioned

Increased Accessibility:

- **Broader Reach:** Makes advanced tools more accessible to a wider audience.

Enhanced Productivity:

- **Efficiency Gains:** Can lead to significant productivity improvements in various technical tasks.

Potential Negative Impacts

Potential Misuse:

- **Malware Creation:** Risk of tools like SWE-Agent being used to develop harmful or malicious code.

Impact on Employment:

- **Programming Jobs:** Potential for displacement or transformation of programming roles due to automation.

Unresolved Issues:

- **Production Environments:**
  Unclear if SWE-Agent faces common issues such as hallucinations.
  Unclear if SWE-Agent produces consistently high quality code.
  (gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality)