P4.3-576

Evaluating AI Code Assistants for Enhanced Software Development in CTBTO: A Modular Approach

Evangelos Dellis

Preparatory Commission for the Comprehensive Nuclear-Test-Ban Treaty Organization (CTBTO),
P.O. Box 1200, 1400 Vienna, Austria

CTBTO PREPARATORY COMMISSION | PUTTING AN END TO NUCLEAR EXPLOSIONS

INTRODUCTION AND MAIN RESULTS

AI code assistants have emerged as a promising solution, offering capabilities such as autocomplete, code refactoring and context-aware suggestions. However, evaluating the effectiveness of these AI-powered tools in the CTBTO context is crucial to ensure their reliability, security, and compliance with organizational standards.

In this work we present the results of the evaluation of open-source AI code assistants in CTBTO, focusing on a modular approach that incorporates multiple components, including autocomplete models, chat models, local and remote context engines, filtering mechanisms, and training engines.

Evaluating AI Code Assistants for Enhanced Software Development in CTBTO: A Modular Approach

Evangelos Dellis

P4.3-576

Introduction and defitinions

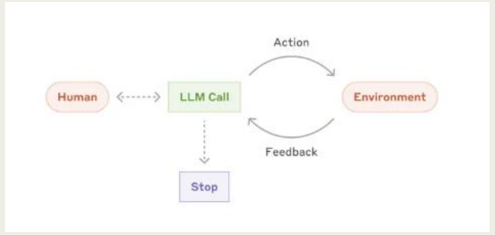
An AI coding Assistant/Agent is a **tool** that uses a Large Language Model (**LLM**) that is finetuned for coding tasks and for “tool-use”, more particularly:

AI Coding Assistants (higher predictability – lower agency)**:**

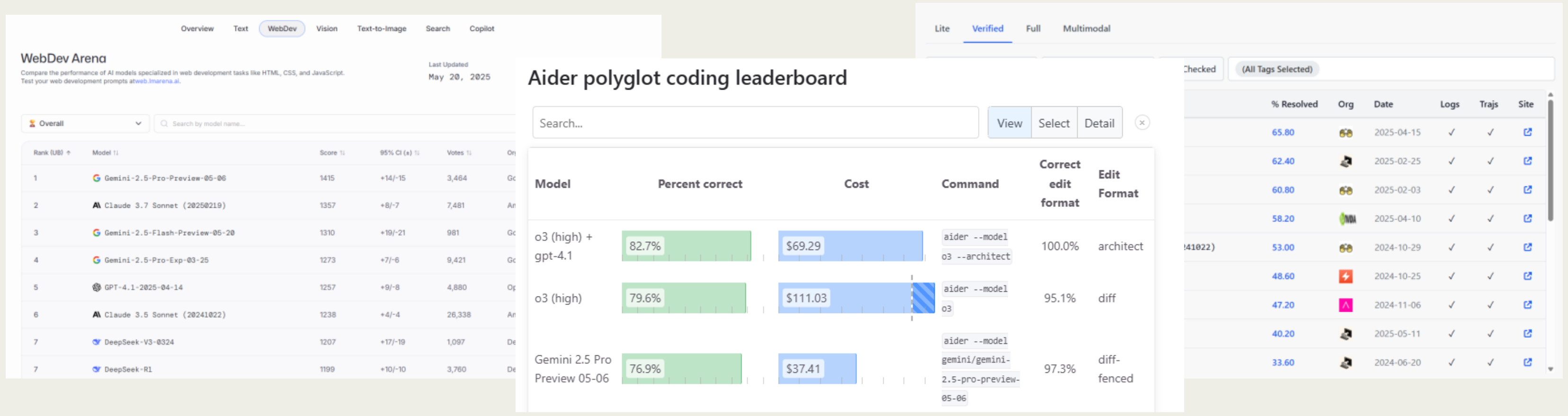
- Allow developers to work faster by automating repetitive or time-consuming tasks
- Solve simple, pre-defined tasks but struggle with novel or open-ended problems
- **Uses an IDE** where the developer is driving the coding

AI Coding Agents (higher agency – lower predictability):

- Agents are typically implemented as an LLM performing actions (via tool-calling) based on environmental feedback **in a loop**.
- Handle complex, multi-step problem-solving **tasks**, often across dynamic environments.



Benchmarks and Leaderboards



Overview Text WebDev Vision Text-to-Image Search Copilot

WebDev Arena

Compare the performance of AI models specialized in web development tasks like HTML, CSS, and JavaScript. Test your web development prompts atweb.lmarena.ai.

Last Updated May 20, 2025

Overall

Rank (UB)	Model	Score	95% CI (±)	Votes
1	Gemini-2.5-Pro-Preview-05-06	1415	+14/-15	3,464
2	Claude 3.7 Sonnet (20250219)	1357	+8/-7	7,481
3	Gemini-2.5-Flash-Preview-05-20	1310	+19/-21	981
4	Gemini-2.5-Pro-Exp-03-25	1273	+7/-6	9,421
5	GPT-4.1-2025-04-14	1257	+9/-8	4,880
6	Claude 3.5 Sonnet (20241022)	1238	+4/-4	26,338
7	DeepSeek-V3-0324	1207	+17/-19	1,097
7	DeepSeek-R1	1199	+10/-10	3,760

Aider polyglot coding leaderboard

Model	Percent correct	Cost	Command	Correct edit format	Edit Format
o3 (high) + gpt-4.1	82.7%	$69.29	aider --model o3 --architect	100.0%	architect
o3 (high)	79.6%	$111.03	aider --model o3	95.1%	diff
Gemini 2.5 Pro Preview 05-06	76.9%	$37.41	aider --model gemini/gemini-2.5-pro-preview-05-06	97.3%	diff-fenced

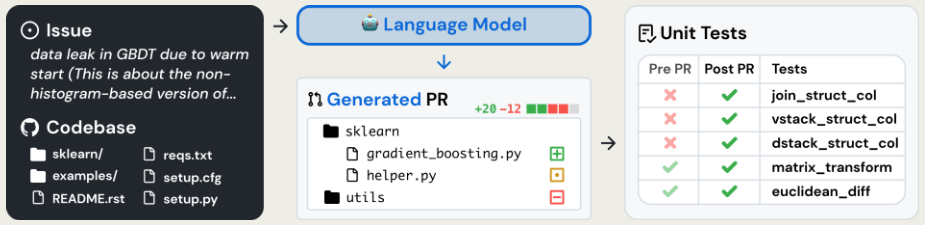
Lite Verified Full Multimodal

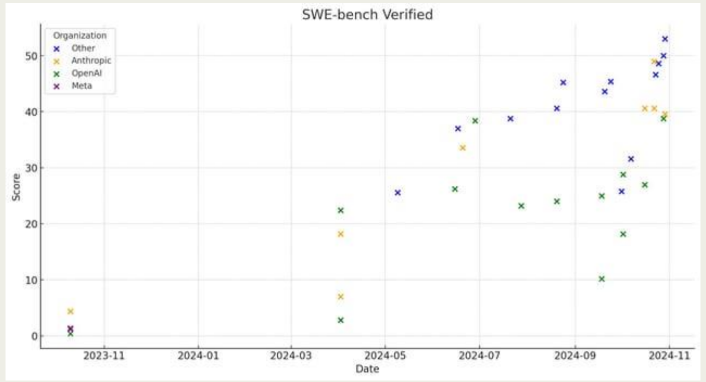
(All Tags Selected)

% Resolved	Date	Logs	Trajs
65.80	2025-04-15	✓	✓
62.40	2025-02-25	✓	✓
60.80	2025-02-03	✓	✓
58.20	2025-04-10	✓	✓
53.00	2024-10-29	✓	✓
48.60	2024-10-25	✓	✓
47.20	2024-11-06	✓	✓
40.20	2025-05-11	✓	✓
33.60	2024-06-20	✓	✓

SWE-bench: A benchmark used to evaluate the ability of large language models (LLMs) to solve real-world software engineering problems

- Given a codebase and an issue, a language model is tasked with generating a patch that **resolves** the issue.
- Results are **verifiable**!





Benchmarks: SWE-bench, Codeforces, HumanEval, MBPP, LiveBench Coding, LiveCodeBench & SciCode

Evaluating AI Code Assistants for Enhanced Software Development in CTBTO: A Modular Approach

Evangelos Dellis

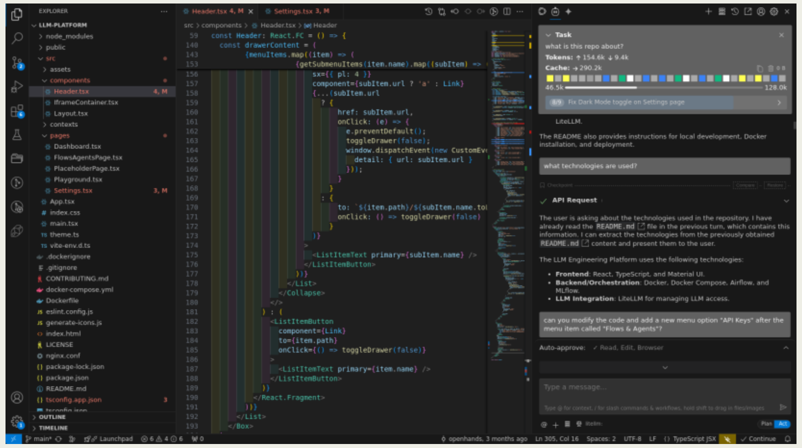
P4.3-576

AI IDE Coding Assistants

An **AI IDE** integrates code-generating LLMs directly into your existing development environment (like VS Code or JetBrains), helping you write, refactor, and understand code.

- Designed to **assist** rather than replace developers: code suggestions, completion, inline explanations, and debugging hints.
- Runs locally or via plugin, with real-time code context and seamless integration into your workflow.

We use vLLM for inference which is suitable for multiuser environments. We installed the SOTA open-source coding LLM for copilots (codestral:22b) locally in our GPU infrastructure.



```
const Header: React.FC = () => {
  const drawerContent = (
      {menuItems.map((item) => (
                {getSubmenuItems(item.name).map((subItem) =>
                  sx={{ pl: 4 }}
                  component={subItem.url ? 'a' : Link}
                  {...(subItem.url
                    ? {
                        href: subItem.url,
                        onClick: (e) => {
                          e.preventDefault();
                          toggleDrawer(false);
                          window.dispatchEvent(new CustomEv
                            detail: { url: subItem.url }
                          }));
                        }
                      }
                    : {
                        to: `${item.path}/${subItem.name.to
                        onClick: () => toggleDrawer(false)
                      }
                  )}
                >
                  <ListItemText primary={subItem.name} />
                </ListItemButton>
              ))}
            </List>
          </Collapse>
        </>
      ) : (
        <ListItemButton
          component={Link}
          to={item.path}
          onClick={() => toggleDrawer(false)}
        >
          <ListItemText primary={item.name} />
        </ListItemButton>
      )}
    </React.Fragment>
  ))}
  </List>
</Box>
```

Task
what is this repo about?
Tokens: ↑ 154.6k ↓ 9.4k
Cache: → 290.2k
46.5k 128.0k
8/9 Fix Dark Mode toggle on Settings page

LiteLLM.

The README also provides instructions for local development, Docker installation, and deployment.

what technologies are used?

API Request

The user is asking about the technologies used in the repository. I have already read the `README.md` file in the previous turn, which contains this information. I can extract the technologies from the previously obtained `README.md` content and present them to the user.

The LLM Engineering Platform uses the following technologies:

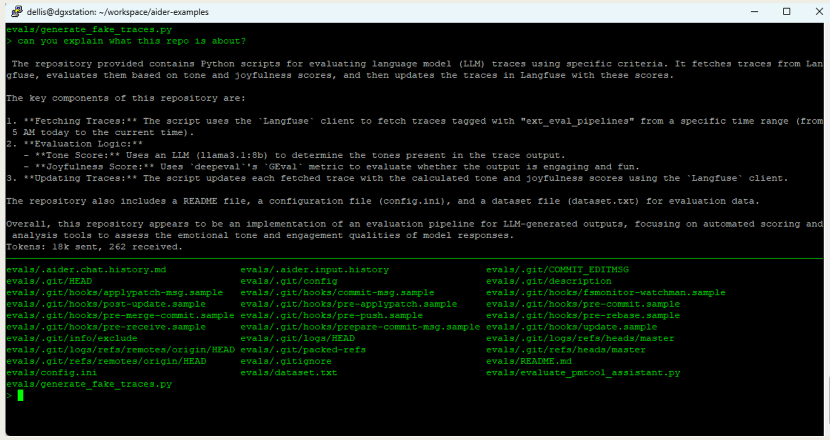
- **Frontend**: React, TypeScript, and Material UI.
- **Backend/Orchestration**: Docker, Docker Compose, Airflow, and MLflow.
- **LLM Integration**: LiteLLM for managing LLM access.

can you modify the code and add a new menu option "API Keys" after the menu item called "Flows & Agents"?

Auto-approve: ✓ Read, Edit, Browser

Type a message...

CLI-based Agents



```
evals/generate_fake_traces.py
> can you explain what this repo is about?

 The repository provided contains Python scripts for evaluating language model (LLM) traces using specific criteria. It fetches traces from Lan
gfuse, evaluates them based on tone and joyfulness scores, and then updates the traces in Langfuse with these scores.

The key components of this repository are:

1. **Fetching Traces:** The script uses the `Langfuse` client to fetch traces tagged with "ext_eval_pipelines" from a specific time range (from
 5 AM today to the current time).
2. **Evaluation Logic:**
   - **Tone Score:** Uses an LLM (llama3.1:8b) to determine the tones present in the trace output.
   - **Joyfulness Score:** Uses `deepeval`'s `GEval` metric to evaluate whether the output is engaging and fun.
3. **Updating Traces:** The script updates each fetched trace with the calculated tone and joyfulness scores using the `Langfuse` client.

The repository also includes a README file, a configuration file (config.ini), and a dataset file (dataset.txt) for evaluation data.

Overall, this repository appears to be an implementation of an evaluation pipeline for LLM-generated outputs, focusing on automated scoring and
 analysis tools to assess the emotional tone and engagement qualities of model responses.
Tokens: 18k sent, 262 received.
────────────────────────────────────────────────────────────────────────────────────────────────────────
evals/.aider.chat.history.md            evals/.aider.input.history                evals/.git/COMMIT_EDITMSG
evals/.git/HEAD                         evals/.git/config                         evals/.git/description
evals/.git/hooks/applypatch-msg.sample  evals/.git/hooks/commit-msg.sample        evals/.git/hooks/fsmonitor-watchman.sample
evals/.git/hooks/post-update.sample     evals/.git/hooks/pre-applypatch.sample    evals/.git/hooks/pre-commit.sample
evals/.git/hooks/pre-merge-commit.sample evals/.git/hooks/pre-push.sample         evals/.git/hooks/pre-rebase.sample
evals/.git/hooks/pre-receive.sample     evals/.git/hooks/prepare-commit-msg.sample evals/.git/hooks/update.sample
evals/.git/info/exclude                 evals/.git/logs/HEAD                      evals/.git/logs/refs/heads/master
evals/.git/logs/refs/remotes/origin/HEAD evals/.git/packed-refs                   evals/.git/refs/heads/master
evals/.git/refs/remotes/origin/HEAD     evals/.gitignore                          evals/README.md
evals/config.ini                        evals/dataset.txt                         evals/evaluate_pmtool_assistant.py
evals/generate_fake_traces.py
>
```

CLI-Based Agents are agents you interact with through your terminal.

- They wrap LLM reasoning into a command-line interface, helping you navigate codebases, answer questions, or modify code—all from within your local dev environment.

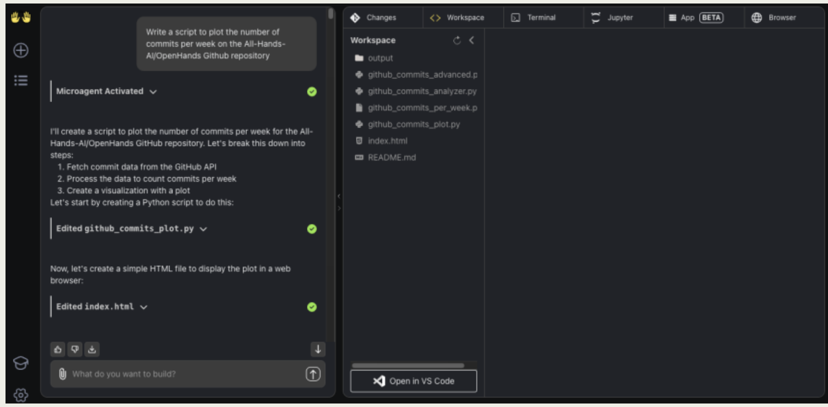
Examples: Claude Code, Codex CLI (open-source), Qwen Code and Aider (open-source)

Both Aider and Codex CLI can work with local docs, codebase from GitHub/Gitlab, can consult documentation, connect to Jira, etc.

SWE-Agent or Teammate Agents

A SWE-Agent is an **asynchronous** software engineering agent that you can **delegate tasks** to and receive a fully tested PR ready to be reviewed and merged in main branch. Examples include Devin, Codex, Jules, **OpenHands** (open-source)

- It is a cloud-based software engineering agent that is running on **public** or **private** cloud (sandboxed) → <u>spins-up new docker containers</u> in the cloud!
- The task can be anything from fixing a bug, review code, do refactors, and implement new functionalities in response to user feedback.
- Integration with Github/Gitlab: ability to clone the repo and the ability to push a pull request



Write a script to plot the number of commits per week on the All-Hands-AI/OpenHands Github repository

Microagent Activated

I'll create a script to plot the number of commits per week for the All-Hands-AI/OpenHands GitHub repository. Let's break this down into steps:
1. Fetch commit data from the GitHub API
2. Process the data to count commits per week
3. Create a visualization with a plot
Let's start by creating a Python script to do this:

Edited `github_commits_plot.py`

Now, let's create a simple HTML file to display the plot in a web browser:

Edited `index.html`

What do you want to build?

CTBTO PREPARATORY COMMISSION | PUTTING AN END TO NUCLEAR EXPLOSIONS

SnT 2025 CTBT: SCIENCE AND TECHNOLOGY CONFERENCE
8 SEPTEMBER ONLINE DAY
9 TO 12 SEPTEMBER AT HOFBURG PALACE, VIENNA & ONLINE

Evaluating AI Code Assistants for Enhanced Software Development in CTBTO: A Modular Approach

Evangelos Dellis

P4.3-576

A guide on how to use an autonomous SWE-Agent

You specify a prompt, and the agent goes to work in its own environment and after a few minutes, the agent gives you back a diff. The environment may contain:

- Set package versions (like python 3.12)
- Environment variables and Secrets
- Setup scripts (pip install -r requirements.txt)

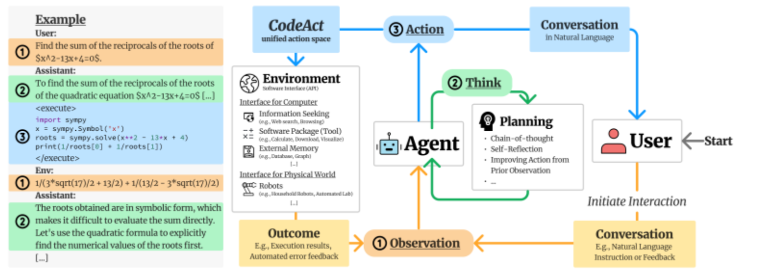
You can execute tasks/prompts in either ask mode or code mode:

a) **Ask mode:** the agent clones a read-only version of your repo, booting faster and giving you follow-up tasks → Ask mode for brainstorming, audits, or architecture questions.
b) **Code mode:** the agent creates a full-fledged environment that the agent can run and test against. → Code mode for when you want automated refactors, tests, or fixes applied

Development Workflow:

1. Connect to GitHub/Gitlab **repo** and select a branch
2. Submit a **new task** (implement a new feature, fix a bug, do refactor, review code, etc)
3. A new **Docker container** based upon a base image is created. The repo is cloned at the desired branch and run any setup scripts you have from the specified working directory (install dependencies, compile, required software to run unit tests, etc).
4. The agent then **runs terminal commands** in a loop. It writes code, runs tests, and attempts to check its work. The agent attempts to honor any specified lint or test commands you've defined in an AGENTS.md file. The agent does not have access to any special tools outside of the terminal or CLI tools you provide.
5. When the agent completes your task, it presents a diff or a set of follow-up tasks. You can choose to **open a PR** or respond with follow-up comments to ask for more changes → a senior developer will review/accept the PR.

Conclusions


Example
User:
① Find the sum of the reciprocals of the roots of $x^2-13x+4=0$.
Assistant:
② To find the sum of the reciprocals of the roots of the quadratic equation $x^2-13x+4=0$ [...]
```
<execute>
import sympy
x = sympy.Symbol('x')
roots = sympy.solve(x**2 - 13*x + 4)
print(1/roots[0] + 1/roots[1])
</execute>
```
Env:
① 1/(3*sqrt(17)/2 + 13/2) + 1/(13/2 - 3*sqrt(17)/2)
Assistant:
② The roots obtained are in symbolic form, which makes it difficult to evaluate the sum directly. Let's use the quadratic formula to explicitly find the numerical values of the roots first.
[...]
CodeAct
unified action space
③ Action
Conversation
in Natural Language
Environment
Software Interface (API)
Interface for Computer
Information Seeking (e.g., Web search, Browsing)
Software Package (Tool) (e.g., Calculate, Download, Visualize)
External Memory (e.g., Database, Graph)
[...]
Interface for Physical World
Robots (e.g., Household Robots, Automated Lab)
[...]
② Think
Agent
Planning
Chain-of-thought
Self-Reflection
Improving Action from Prior Observation
...
User
Start
Initiate Interaction
Outcome
E.g., Execution results, Automated error feedback
① Observation
Conversation
E.g., Natural Language Instruction or Feedback

AI IDEs: We evaluated VS Code Continue vs Cline: Both support autocomplete, code refactor, access to documents, connect to DBs, connect to GitHub/Gitlab, Jira, etc. → augments developer's work (copilot)

CLI-based Agents: Both tools can work with local docs, Gitlab codebases, can consult documentation, connect to Jira, etc → good Linux CLI skills are crucial

Teammate Agents (SWE-Agent): SWE-Agents are experimental coding agents that helps you fix bugs and build new features. They integrate with GitHub/Gitlab, understands your codebase, and works asynchronously. We evaluated OpenHands with different LLMs and for different scenarios → excels in greenfield development!

CTBTO PREPARATORY COMMISSION
PUTTING AN END TO NUCLEAR EXPLOSIONS