THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY | DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

RELAXED SYSTEM LAB

LLM Agent

COMP4901Y

Binhang Yuan

RELAXED
SYSTEM LAB

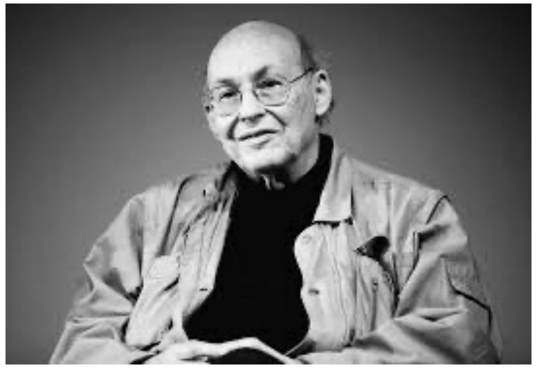
AI Agent

A Little Background about AI

RELAXED SYSTEM LAB

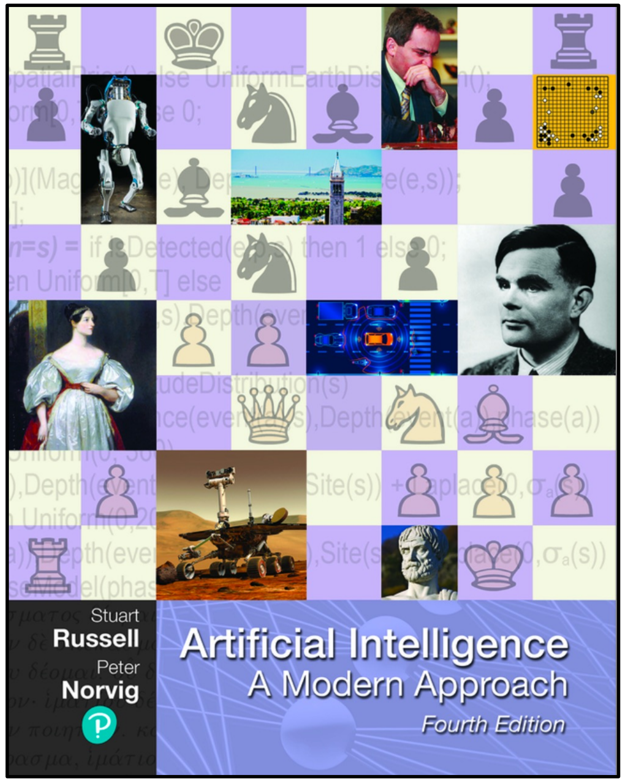
- What is AI?
- *"The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning …"* – Richard Bellman

- *"AI is the science of making machines do things that would require intelligence if done by men."* – Marvin Minsky



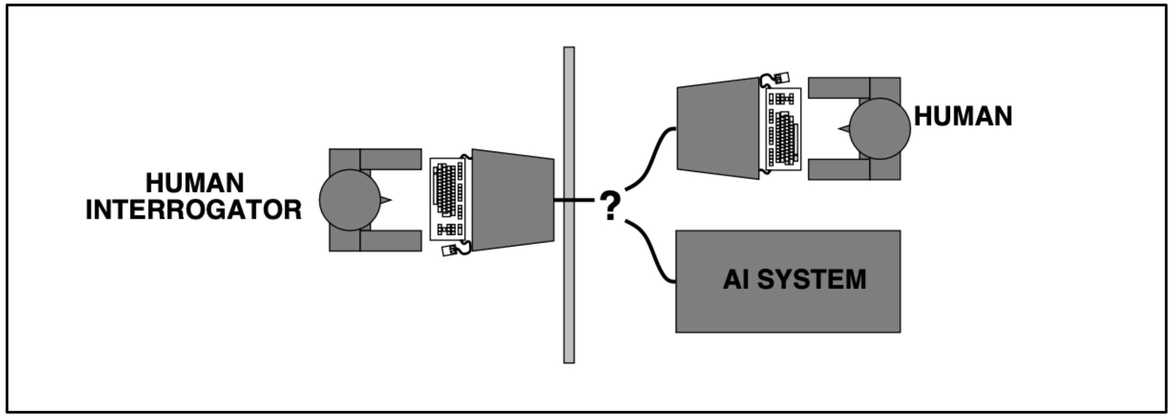
A Little Background about AI

- What is AI?
- Four possible combinations from two dimensions:
  - Human vs. Rational
  - Thought vs. Behaviour
- The view of AI falls into four categories:
  - Acting humanly
  - Thinking humanly
  - Thinking rationally
  - Acting rationally



What is AI? Acting Humanly

- ***The Turing test***:
  - A thought experiment of "Can a machine think?"
  - "Can machines behave intelligently?"
  - A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.
- Problem: Turing test is not reproducible, constructive, or amenable to mathematical analysis.
- Can ChatGPT pass the Turning test?
  - https://humsci.stanford.edu/feature/study-finds-chatgpts-latest-bot-behaves-humans-only-better



What is AI? Thinking Humanly

- ***Cognitive science***:
- Require some scientific theories of internal activities of the brain:
  - What level of abstraction? Knowledge or circuits?
  - How to validate? Requires:
    - Predicting and testing the behavior of human subjects (top-down);
    - or Direct identification from neurological data (bottom-up).
- Both approaches (roughly, Cognitive Science and Cognitive Neuroscience) are now distinct from AI.

What is AI? Thinking Rationally

- ***Law of Thought***:
- Logic:
    - These laws of thought were supposed to govern the operation of the mind;
    - "Socrates is a man" and "all men are mortal" => "Socrates is mortal".
- Probability:
    - Allow rigorous reasoning with uncertain information.
- Problems:
    - Not all intelligent behaviour is mediated by logical deliberation;
    - What is the purpose of thinking? What thoughts should I have?

What is AI? Acting Rationally

- ***A rational agent:***
  - Rational behavior: do the right thing.
  - The right thing is expected to maximize goal achievement, given the available information.
  - Doesn't necessarily involve thinking---e.g., blinking reflex---but thinking should be in the service of rational action.
- Formally, an agent is an entity that perceives and acts, which can be defined as a function from percept histories to actions:
  - $f: P^* \rightarrow A$
- From any given class of environments and tasks, we seek the agent (or class of agents) with the best performance.

Single-Agent vs. Multi-Agent Frameworks

RELAXED SYSTEM LAB

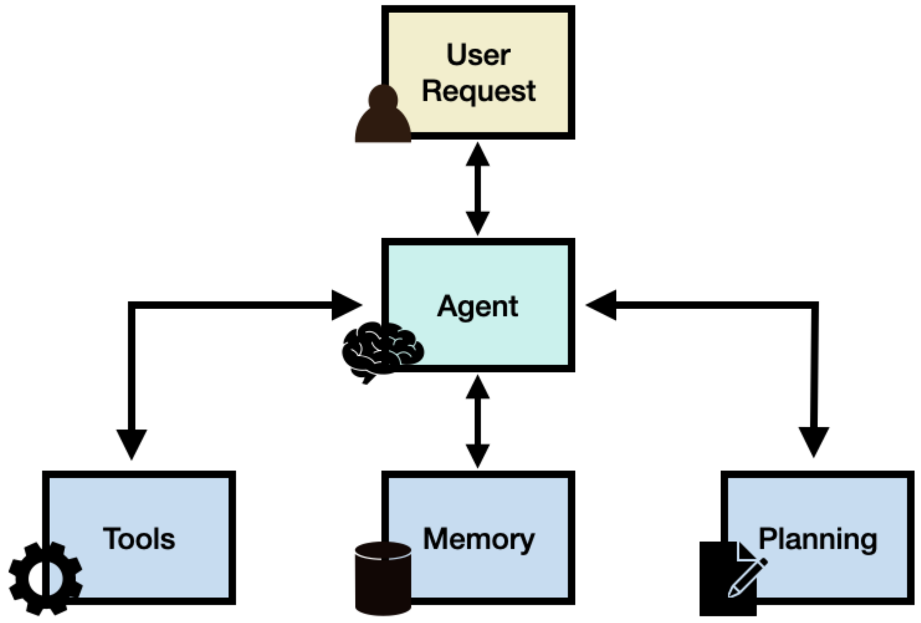
- **Single-Agent Framework**:
  - A single-agent system utilizes one AI entity to manage all tasks, encompassing planning, execution, and decision-making.
- **Multi-Agent Framework**:
  - A multi-agent system comprises multiple AI agents, each specialized in distinct tasks, collaborating to achieve a common goal.

RELAXED SYSTEM LAB

LLM Agent

LLM Agent Framework

- LLM applications that can execute complex tasks:
  - Through the use of LLMs;
  - And other key modules like planning and memory.
- The architecture of an LLM Agent framework can be very flexible, we just provide one simple while clear categorization, which includes:
  - **User Request**: a user question or request;
  - **Agent/Brain**: the agent core acting as coordinator;
  - **Planning**: assists the agent in planning future actions;
  - **Memory**: manages the agent's past behaviours;
  - **Tool**: interacts with external environments.



LLM Agent Framework - Agent

- A large language model (LLM) with general-purpose capabilities serves as the main brain, agent module, or coordinator of the system.
- This component will be activated using a ***prompt template*** that entails important details about how the agent will operate, and the tools it will have access to (along with tool details).
- While not mandatory, an agent can be profiled or be assigned a persona to define its role.
- This profiling information is typically written in the prompt which can include specific details like role details, personality, social information, and other demographic information.

LLM Agent Framework - Planning

RELAXED
SYSTEM LAB

- *Planning without feedback:*
  - Leverage an LLM to decompose a detailed plan;
  - LLM break down the necessary steps or subtasks;
  - Then solve individually to answer the user request.
- *Planning with feedback:*
  - We can leverage a mechanism that enables the model to iteratively reflect and refine the execution plan based on past actions and observations.
  - The goal is to correct and improve on past mistakes which helps to improve the quality of final results.

LLM Agent Framework - Memory

RELAXED SYSTEM LAB

- The memory module helps to store the agent's internal logs, including:
  - past thoughts;
  - past actions;
  - observations from the environment;
  - interactions between agent and user.
- There are also different memory formats to consider when building agents, representative memory formats include:
  - Natural language;
  - Embeddings;
  - Structural data, e.g., relational database.

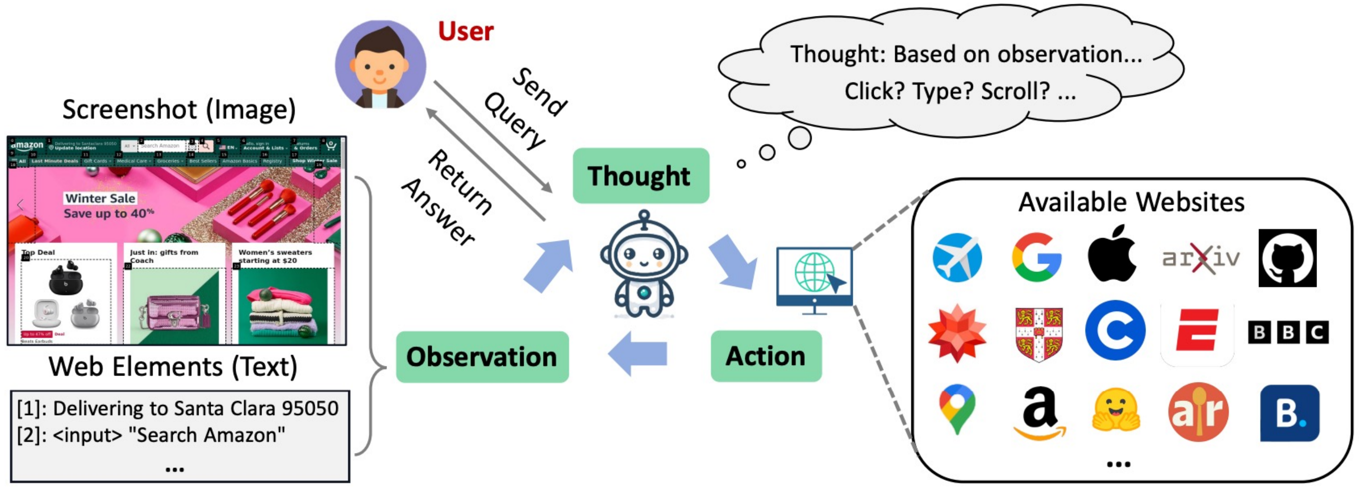
LLM Agent Framework - Tool

- Tools correspond to a set of tool/s that enables the LLM agent to interact with external environments such as:
  - Wikipedia Search API;
  - Code Interpreter;
  - Math Engine;
  - Tools could also include databases, knowledge bases, and external models.
- When the agent interacts with external tools it executes tasks via workflows that assist the agent to obtain observations or necessary information to complete subtasks and satisfy the user request.

RELAXED
SYSTEM LAB

LLM Agent Case Study — WebVoyager

WebVoyager



WebVoyager takes web tasks assigned by a human and automatically browses the web online. At each step, WebVoyager selects actions based on screenshots and text (the 'type' of the web element and its contents). Once the task is completed, the answers will be returned to the user. For example, for a user query: "Find the cost of a 2-year protection for PS4 on Amazon.", the agent interacts with Amazon online, locates the PS4, identifies the 2-year protection price, and returns "$30.99" to the user.

WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models

Hongliang He[1,3*], Wenlin Yao[2], Kaixin Ma[2], Wenhao Yu[2], Yong Dai[2], Hongming Zhang[2], Zhenzhong Lan[3], Dong Yu[2]
[1]Zhejiang University, [2]Tencent AI Lab, [3]Westlake University
hehongliang@westlake.edu.cn, wenlinyao@global.tencent.com

Abstract

The rapid advancement of large language models (LLMs) has led to a new era marked by the development of autonomous applications in real-world scenarios, which drives innovation in creating advanced web agents. Existing web agents typically only handle one input modality and are evaluated only in simplified web simulators or static web snapshots, greatly limiting their applicability in real-world scenarios. To bridge this gap, we introduce WebVoyager, an innovative Large Multimodal Model (LMM) powered web agent that can complete user instructions end-to-end by interacting with real-world websites. Moreover, we establish a new benchmark by compiling real-world tasks from 15 popular websites and introduce an automatic evaluation protocol leveraging multimodal understanding abilities of GPT-4V to evaluate open-ended web agents. We show that WebVoyager achieves a 59.1% task success rate on our benchmark, significantly surpassing the performance of both GPT-4 (All Tools) and the WebVoyager (text-only) setups, underscoring the exceptional capability of WebVoyager. The proposed automatic evaluation metric achieves 85.3% agreement with human judgment, indicating its effectiveness in providing reliable and accurate assessments of web agents.[1]

1 Introduction

The recent advancement of large language models (LLMs), such as ChatGPT and GPT-4 (OpenAI, 2023), have sparked significant interest in developing LLM-based autonomous agents (AutoGPT, 2022) for complex task execution (Qin et al., 2023; Schick et al., 2023). Recent studies have explored the construction of text-based web browsing environments and how to instruct large language model agents to perform web navigation (Nakano et al., 2021; Gur et al., 2023; Zhou et al., 2023; Lu et al., 2023). The primary challenge in these works lies in managing complex and verbose HTML texts, and solutions include simplifying and structuring HTML (Nakano et al., 2021; Zhou et al., 2023; Gur et al., 2023; Deng et al., 2023).

However, existing approaches overlook a critical functionality of browsing: rendering HTML into visual webpages. Particularly, vision capability is crucial for utilizing tools such as web browsers, as rendered web pages are inherently designed with user experience (UX), emphasizing intuitive information and structured presentation. This design principle of rendering makes visual analysis more effective than mere HTML representation. At present, large multimodal models (LMMs), particularly GPT-4V(ision) (OpenAI, 2023) and Gemini (Team et al., 2023), demonstrate a remarkable ability to integrate intricate visual cues with textual information. Existing studies such as Pix2Struct (Lee et al., 2023) and WebArena (Zhou et al., 2023), have initiated explorations into using screenshots as inputs for decision-making in web navigation, yet these are preliminary and do not represent a deep exploration. Therefore, building multimodal web agents to leverage the environment rendered by browsers through screenshots, thus mimicking human web browsing behavior, is now a viable approach to enhance web navigation abilities.

We introduce WebVoyager (Figure 1), a multimodal web agent designed to autonomously accomplish web tasks online from start to finish, managing the entire process end-to-end without any intermediate human intervention. WebVoyager processes the user query by making observations from screenshots and textual content in interactive web elements, formulates a thought on what action to take (such as clicking, typing, or scrolling, etc.), and then executes that action on the websites. Inspired by Set-of-Mark Prompting (Yang et al., 2023a), we mark interactive web elements on screenshots (see Figure 2) to facilitate decision-

*Work done during the internship at Tencent AI Lab.
[1]Our code and data will be released at https://github.com/MinorJerry/WebVoyager

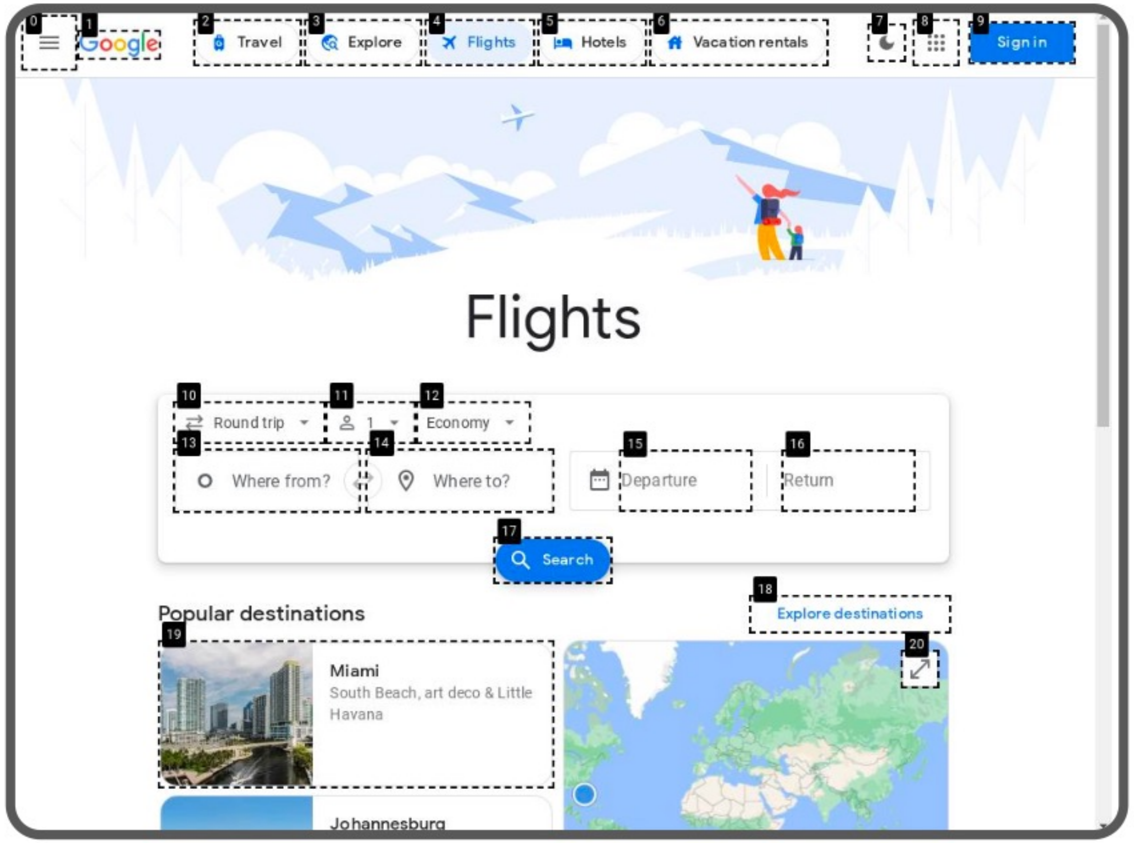
6864

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6864–6890
August 11-16, 2024 ©2024 Association for Computational Linguistics

WebVoyager

Observation Space

- Takes the visual information from the web(screenshots) as the primary source of input;
- Add borders to most of the interactive elements on the web pages and label them with numerical tags in the top left corner.

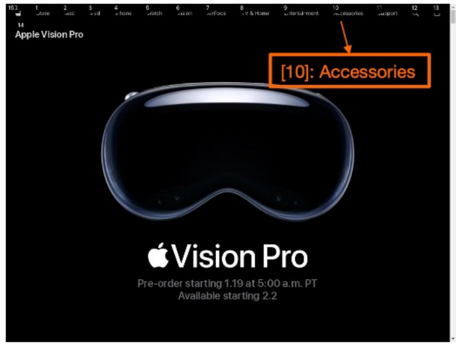


WebVoyager

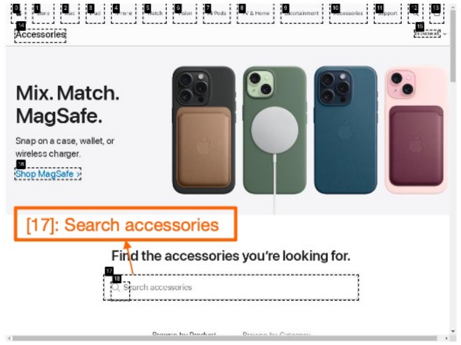
RELAXED SYSTEM LAB

Action Space:

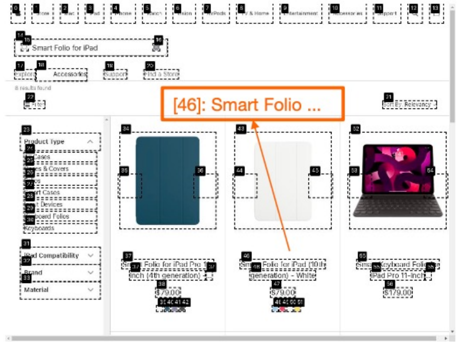
- Click;
- Input;
- Scroll;
- Wait;
- Back;
- Jump to Search Engine;
- Answer.



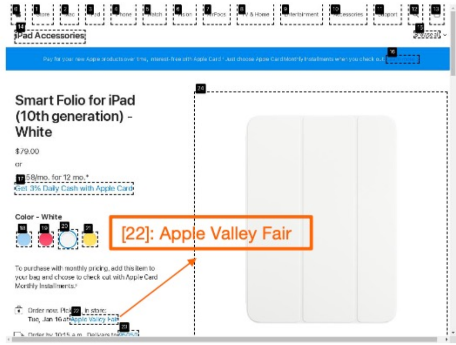
Step 1: Cilck [10]



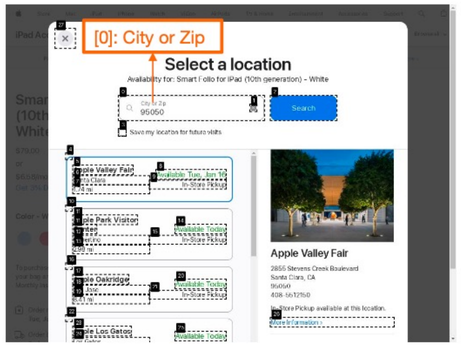
Step 2: Type [17]: Smart Folio for iPad



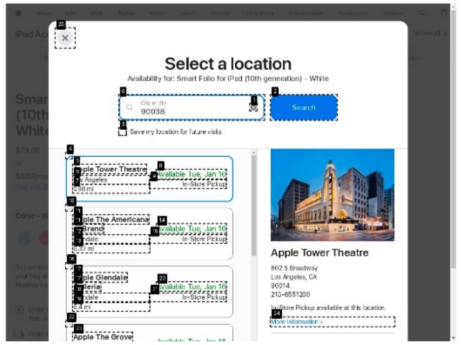
Step 3: Cilck [46]



Step 4: Cilck [22]



Step 5: Type [0]: 90038



Step 6: ANSWER

Given an instruction, WebVoyager:

1. Instantiate a web browser.
2. Perform actions with visual (i.e., screenshots) and textual (i.e., HTML elements) signals from the web.
3. The agent produces an action based on the inputs at every step.
4. The action is then executed in the browser environment.

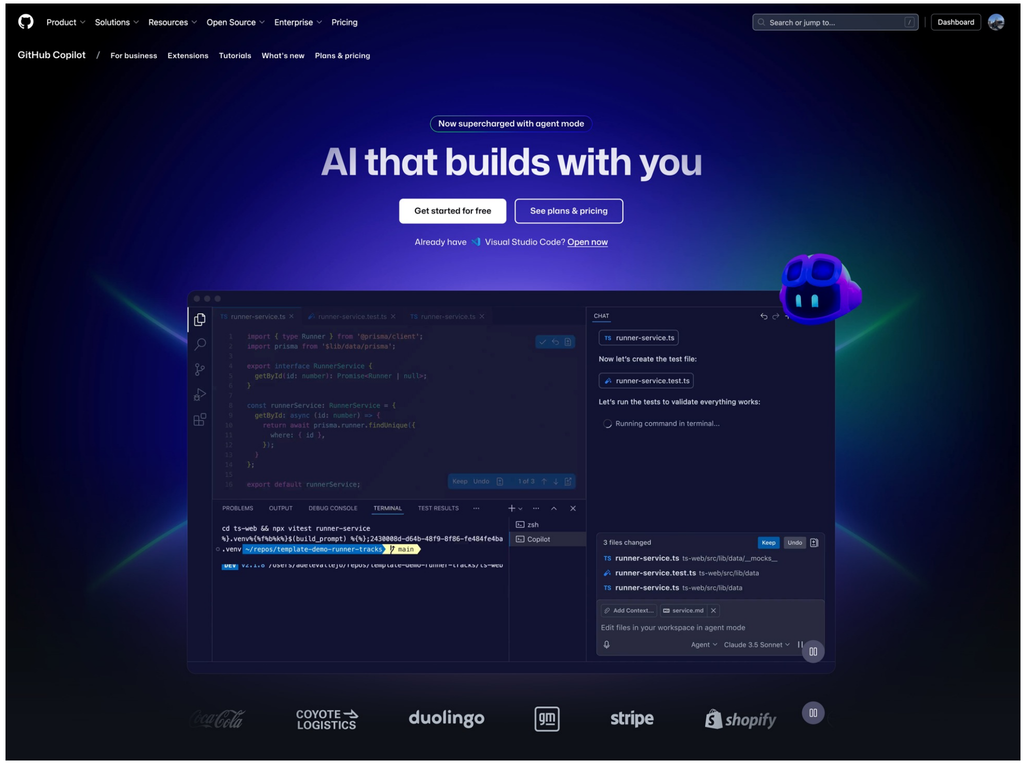
The process continues until the agent decides to stop.

RELAXED
SYSTEM LAB

LLM Agent Case Study — GitHub Copilot

GitHub Copilot: Coding Assistant

- Copilot is an AI pair programmer tool in Visual Studio Code.
- Provide code suggestions as you type in the editor:
  - **AI Code completions**: start typing in the editor, and Copilot provides code suggestions based on your existing code that matches your coding style.
- Use natural language chat to ask about the code;
  - **Natural language chat**: use a conversational interface to ask about your codebase or make edits across your project. Switch between ask, edit, agent mode based on your needs.
- Start an editing session for implementing new feature and fixing bugs.
  - **Smart actions**: boost your developer productivity, from the terminal, source control operations, to debugging and testing code.



```
import { type Runner } from '@prisma/client';
import prisma from '$lib/data/prisma';

export interface RunnerService {
  getById(id: number): Promise<Runner | null>;
}

const runnerService: RunnerService = {
  getById: async (id: number) => {
    return await prisma.runner.findUnique({
      where: { id },
    });
  }
};

export default runnerService;
```

```
cd ts-web && npx vitest runner-service
%}.venv%{%f%b%k%}$(build_prompt) %{%};2430008d-d64b-48f9-8f86-fe484fe4ba
.venv ~/repos/template-demo-runner-tracks main
```

runner-service.ts

Now let's create the test file:

runner-service.test.ts

Let's run the tests to validate everything works:

Running command in terminal...

3 files changed — Keep Undo

runner-service.ts ts-web/src/lib/data/__mocks__

runner-service.test.ts ts-web/src/lib/data

runner-service.ts ts-web/src/lib/data

Add Context... service.md

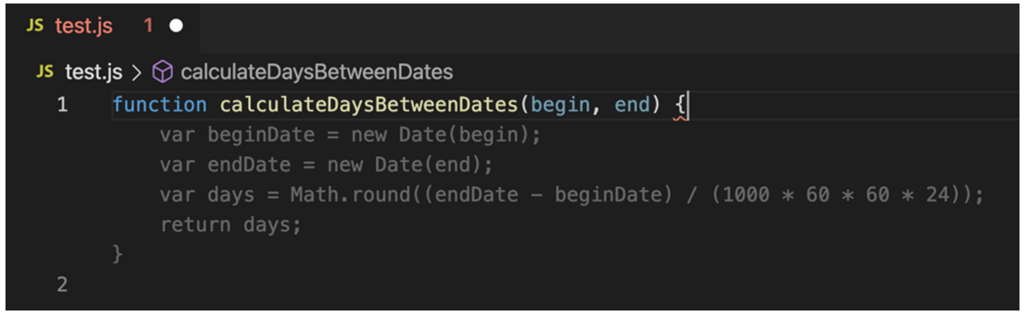
Edit files in your workspace in agent mode

Agent — Claude 3.5 Sonnet

GitHub Copilot: AI Code Completions

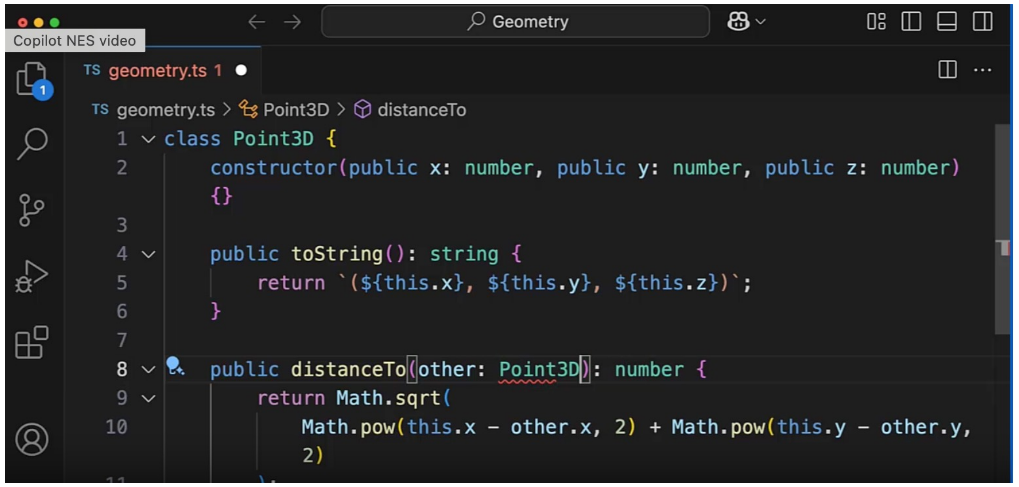
RELAXED SYSTEM LAB

- Two kinds of code completions:
- **<u>Code completions</u>**: Start typing in the editor, and Copilot provides code suggestions that match your coding style and take your existing code into account.
- **<u>Next Edit Suggestions</u>**: Predict your next code edit with Copilot Next Edit Suggestions, aka Copilot NES. Based on the edits you're making, Copilot NES both predicts the location of the next edit you'll want to make and what that edit should be.



```
function calculateDaysBetweenDates(begin, end) {
    var beginDate = new Date(begin);
    var endDate = new Date(end);
    var days = Math.round((endDate - beginDate) / (1000 * 60 * 60 * 24));
    return days;
}
```

Code Completions

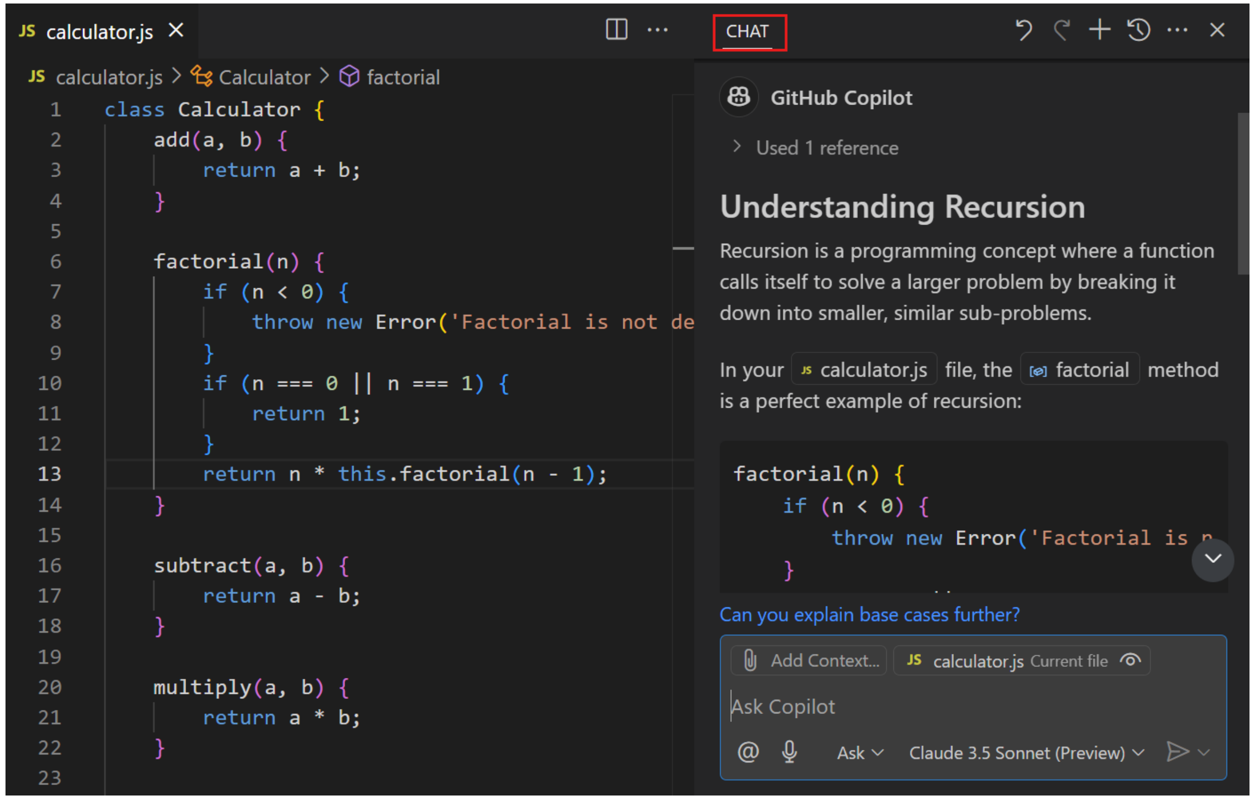


```
class Point3D {
    constructor(public x: number, public y: number, public z: number)
    {}

    public toString(): string {
        return `(${this.x}, ${this.y}, ${this.z})`;
    }

    public distanceTo(other: Point3D): number {
        return Math.sqrt(
            Math.pow(this.x - other.x, 2) + Math.pow(this.y - other.y,
            2)
```

Next Edit Suggestions

GitHub Copilot: Natural Language Chat

- Use chat in VS Code when you need to:
  - **Understand code**: "Explain how this authentication middleware works"
  - **Debug issues**: "Why am I getting a null reference in this loop?"
  - **Get code suggestions**: "Show me how to implement a binary search tree in Python"
  - **Optimize performance**: "Help me improve the efficiency of this database query"
  - **Learn best practices**: "What's the recommended way to handle errors in async functions?"
  - **Get VS Code tips**: "How do I customize keyboard shortcuts?"



```
class Calculator {
    add(a, b) {
        return a + b;
    }

    factorial(n) {
        if (n < 0) {
            throw new Error('Factorial is not de
        }
        if (n === 0 || n === 1) {
            return 1;
        }
        return n * this.factorial(n - 1);
    }

    subtract(a, b) {
        return a - b;
    }

    multiply(a, b) {
        return a * b;
    }

```

Understanding Recursion

Recursion is a programming concept where a function calls itself to solve a larger problem by breaking it down into smaller, similar sub-problems.

In your calculator.js file, the factorial method is a perfect example of recursion:

```
factorial(n) {
    if (n < 0) {
        throw new Error('Factorial is n
    }
```

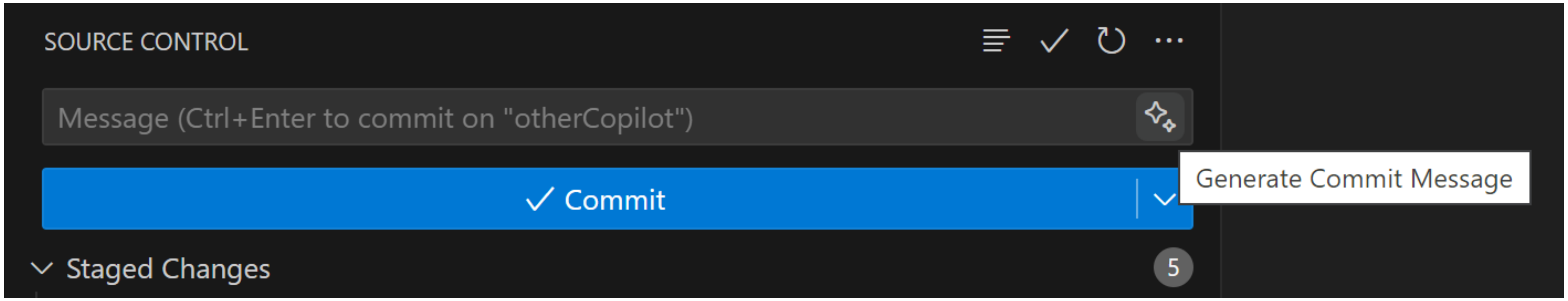
Can you explain base cases further?

Natural Language Chat

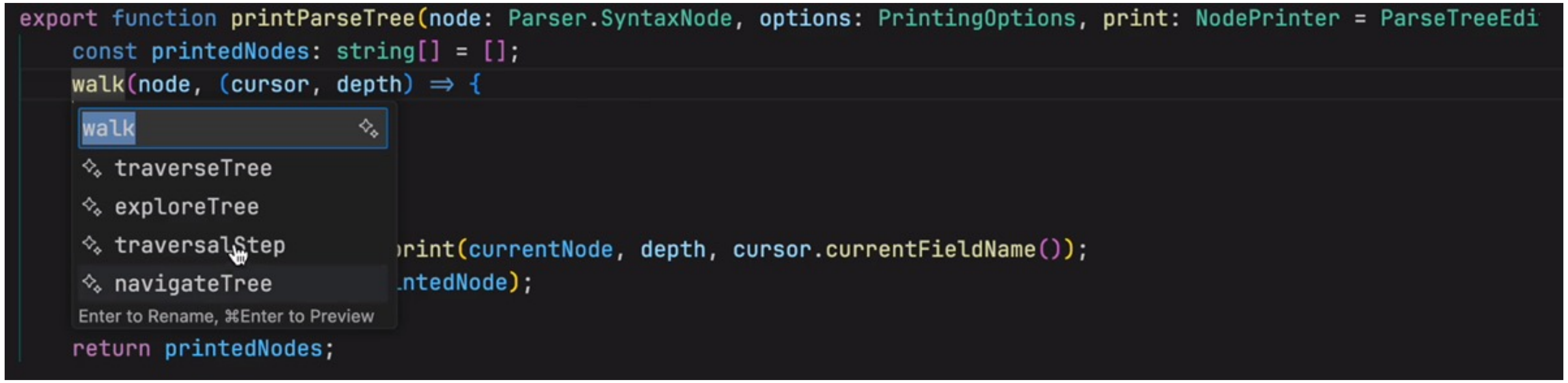
GitHub Copilot: Smart Actions

- For several common scenarios, you can use smart actions to get help from Copilot without having to write a prompt:
  - Generate a commit message and PR information;
  - Rename symbols;
  - Generate documentation;
  - Generate tests;
  - Explain code;
  - Fix coding errors;
  - Fix testing errors;
  - Fix terminal errors;
  - Review code;
  - Semantic search results.

GitHub Copilot: Smart Actions



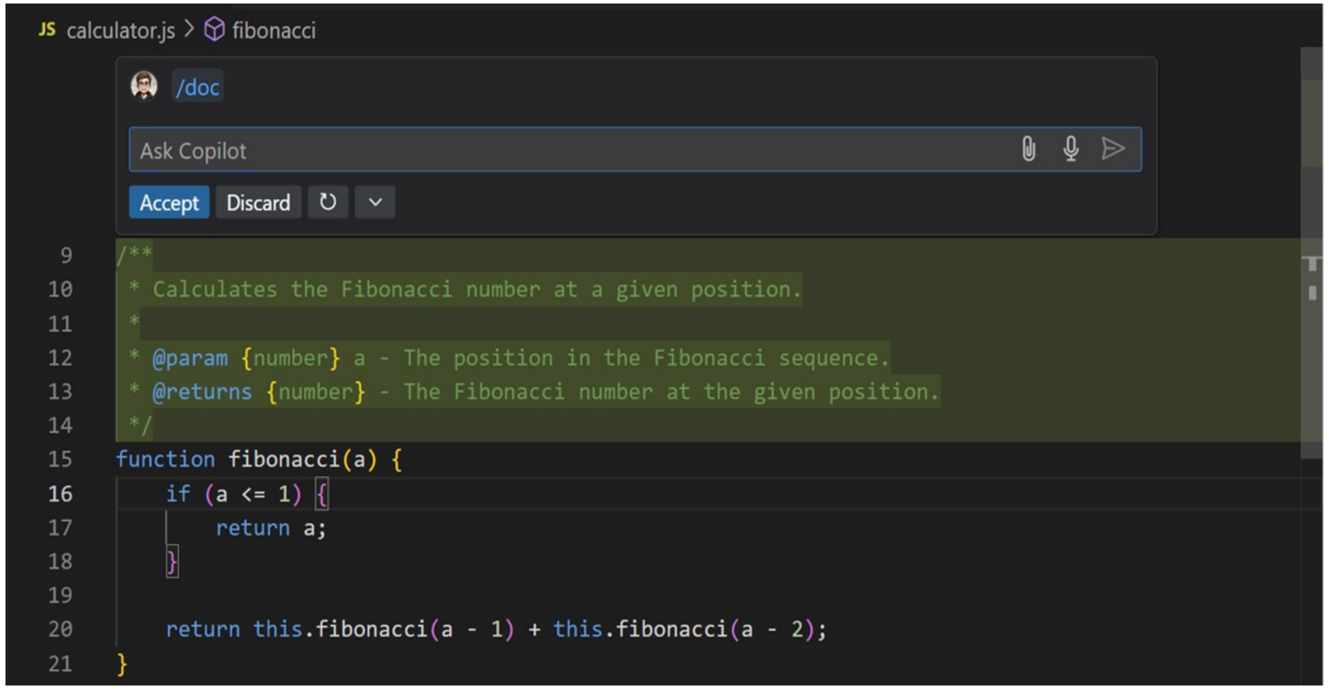
Generate a commit message and PR information



```
export function printParseTree(node: Parser.SyntaxNode, options: PrintingOptions, print: NodePrinter = ParseTreeEdi
    const printedNodes: string[] = [];
    walk(node, (cursor, depth) ⇒ {
      walk
      traverseTree
      exploreTree
      traversalStep                  rint(currentNode, depth, cursor.currentFieldName());
      navigateTree                   ntedNode);
      Enter to Rename, ⌘Enter to Preview
    return printedNodes;
```

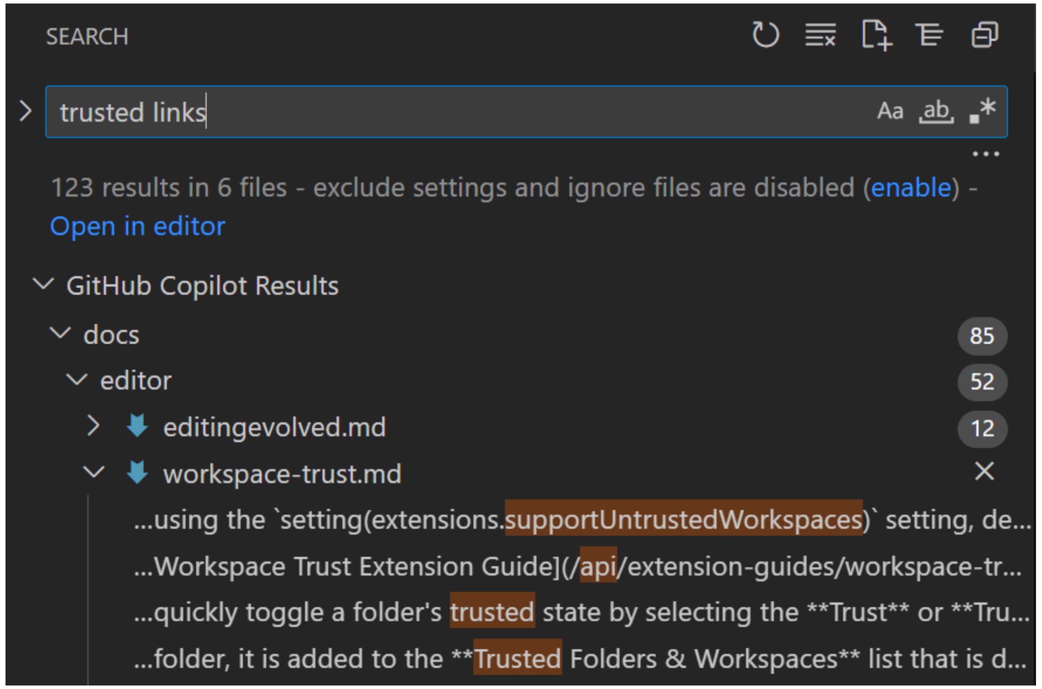
Rename symbols

GitHub Copilot: Smart Actions



```
/**
 * Calculates the Fibonacci number at a given position.
 *
 * @param {number} a - The position in the Fibonacci sequence.
 * @returns {number} - The Fibonacci number at the given position.
 */
function fibonacci(a) {
    if (a <= 1) {
        return a;
    }

    return this.fibonacci(a - 1) + this.fibonacci(a - 2);
}
```

Generate documentation



123 results in 6 files - exclude settings and ignore files are disabled (enable) - Open in editor

...using the `setting(extensions.supportUntrustedWorkspaces)` setting, de...
...Workspace Trust Extension Guide](/api/extension-guides/workspace-tr...
...quickly toggle a folder's trusted state by selecting the **Trust** or **Tru...
...folder, it is added to the **Trusted Folders & Workspaces** list that is d...

Semantic search results

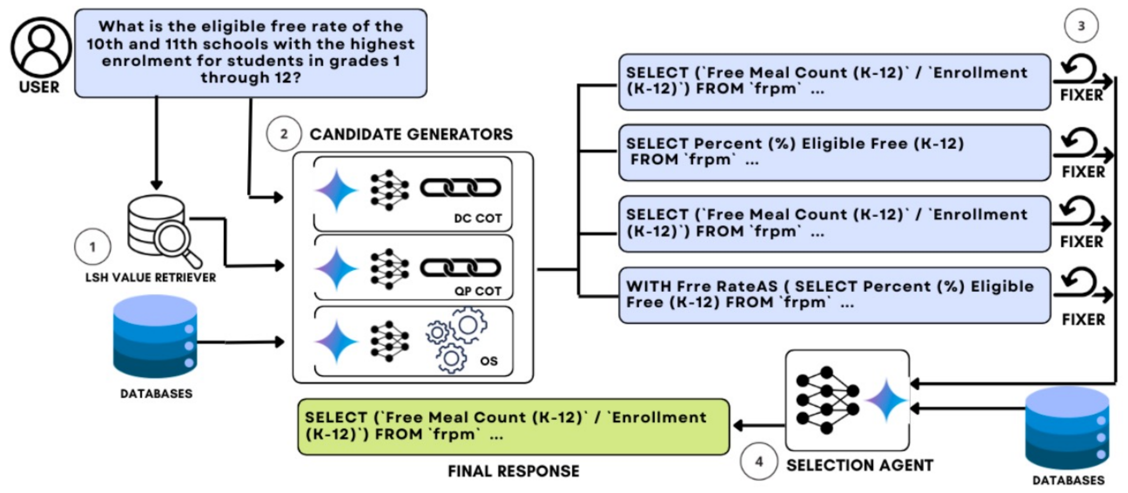
LLM Agent Case Study — CHASE-SQL

CHASE-SQL
— An Agent-Based Approach to Text-to-SQL

RELAXED SYSTEM LAB

CHASE-SQL (a Multi-Agent Framework) breaks the task into modular sub-tasks, generate SQL, fix errors, and select the best result.

- 1. Value retrieval: Extract keywords from the given question using an LLM prompted. For each keyword, employ some measurement, i.e., locality-sensitive hashing, to retrieve the most syntactically-similar words in database schema.
- 2. Candidate generator:
  - **Planner**: breaks the question into smaller sub-questions that are easier to solve. Each sub-question can be translated into a partial SQL (i.e., an intermediate step);
  - **SQL synthesizer**: An LLM-based agent that actually composes SQL statements (candidates) based on the plan and retrieved info.



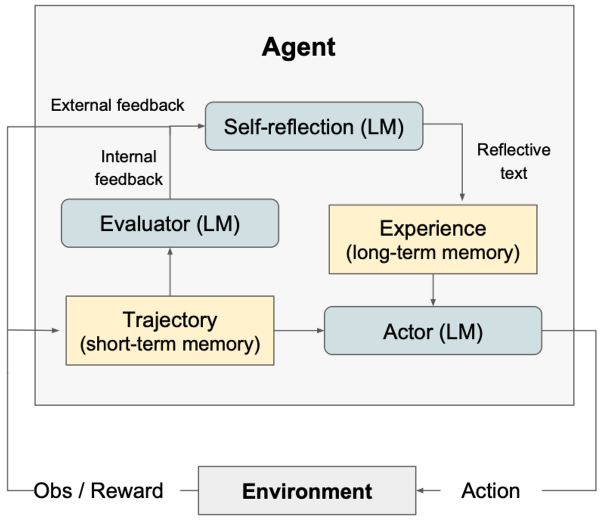
```
SELECT (`Free Meal Count (K-12)` / `Enrollment (K-12)`) FROM `frpm` ...
SELECT Percent (%) Eligible Free (K-12) FROM `frpm` ...
SELECT (`Free Meal Count (K-12)` / `Enrollment (K-12)`) FROM `frpm` ...
WITH Frre RateAS ( SELECT Percent (%) Eligible Free (K-12) FROM `frpm` ...
```

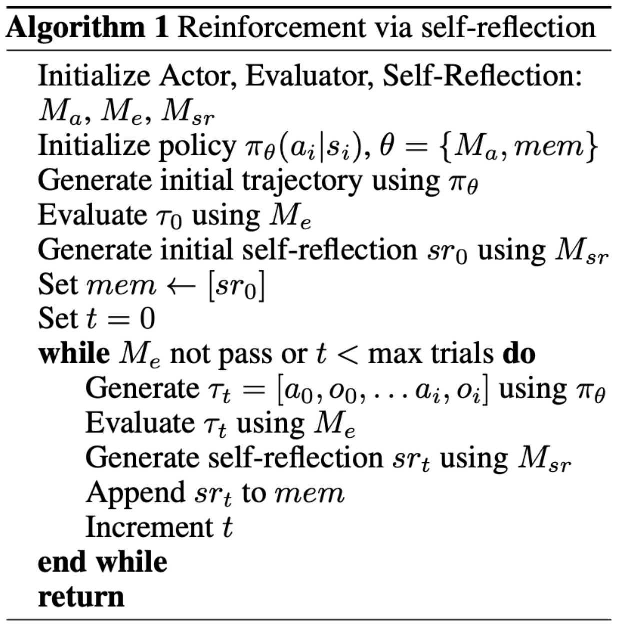
```
SELECT (`Free Meal Count (K-12)` / `Enrollment (K-12)`) FROM `frpm` ...
```

- 3. **Query fixer**: *“self-reflects”* on the previously generated query, using feedback such as syntax error details or empty result sets to guide the correction process
- 4. **Selection agent**: Choose the best SQL query from the set of viable candidates.

Self-Reflection in LLM Agent

- An **Actor** model generates text and actions conditioned on the state observations;
- An **Evaluator** model assesses the quality of the generated outputs produced by the Actor;
- An **Self-Reflection** model generates verbal reinforcement cues to assist the Actor in self-improvement.




Algorithm 1 Reinforcement via self-reflection

```
Initialize Actor, Evaluator, Self-Reflection:
$M_a$, $M_e$, $M_{sr}$
Initialize policy $\pi_\theta(a_i|s_i)$, $\theta = \{M_a, mem\}$
Generate initial trajectory using $\pi_\theta$
Evaluate $\tau_0$ using $M_e$
Generate initial self-reflection $sr_0$ using $M_{sr}$
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while $M_e$ not pass or $t <$ max trials do
    Generate $\tau_t = [a_0, o_0, \dots a_i, o_i]$ using $\pi_\theta$
    Evaluate $\tau_t$ using $M_e$
    Generate self-reflection $sr_t$ using $M_{sr}$
    Append $sr_t$ to $mem$
    Increment $t$
end while
return
```

Reflexion: Language Agents with Verbal Reinforcement Learning

Noah Shinn
Northeastern University
noahshinn024@gmail.com

Federico Cassano
Northeastern University
cassano.f@northeastern.edu

Edward Berman
Northeastern University
berman.ed@northeastern.edu

Ashwin Gopinath
Massachusetts Institute of Technology
agopi@mit.edu

Karthik Narasimhan
Princeton University
karthikn@princeton.edu

Shunyu Yao
Princeton University
shunyuy@princeton.edu

Abstract

Large language models (LLMs) have been increasingly used to interact with external environments (e.g., games, compilers, APIs) as goal-driven agents. However, it remains challenging for these language agents to quickly and efficiently learn from trial-and-error as traditional reinforcement learning methods require extensive training samples and expensive model fine-tuning. We propose *Reflexion*, a novel framework to reinforce language agents not by updating weights, but instead through linguistic feedback. Concretely, Reflexion agents verbally reflect on task feedback signals, then maintain their own reflective text in an episodic memory buffer to induce better decision-making in subsequent trials. Reflexion is flexible enough to incorporate various types (scalar values or free-form language) and sources (external or internally simulated) of feedback signals, and obtains significant improvements over a baseline agent across diverse tasks (sequential decision-making, coding, language reasoning). For example, Reflexion achieves a 91% pass@1 accuracy on the HumanEval coding benchmark, surpassing the previous state-of-the-art GPT-4 that achieves 80%. We also conduct ablation and analysis studies using different feedback signals, feedback incorporation methods, and agent types, and provide insights into how they affect performance. We release all code, demos, and datasets at https://github.com/noahshinn024/reflexion.

1 Introduction

Recent works such as ReAct [30], SayCan [1], Toolformer [22], HuggingGPT [23], generative agents [19], and WebGPT [17] have demonstrated the feasibility of autonomous decision-making agents that are built on top of a large language model (LLM) core. These methods use LLMs to generate text and 'actions' that can be used in API calls and executed in an environment. Since they rely on massive models with an enormous number of parameters, such approaches have been so far limited to using in-context examples as a way of teaching the agents, since more traditional optimization schemes like reinforcement learning with gradient descent require substantial amounts of compute and time.

Preprint. Under review.

References

- https://github.com/pemagrg1/AI_class2022/blob/main/book/Artificial-Intelligence-A-Modern-Approach-4th-Edition-1-compressed.pdf
- https://www.promptingguide.ai/research/llm-agents
- https://code.visualstudio.com/docs/copilot/ai-powered-suggestions
- https://code.visualstudio.com/docs/copilot/chat/copilot-chat
- https://code.visualstudio.com/docs/copilot/copilot-smart-actions
- https://arxiv.org/abs/2410.01943
- https://angelxuanchang.github.io/nlp-class/assets/lecture-slides/L20-LLM-agents.pdf
- https://arxiv.org/abs/2210.03629
- https://aclanthology.org/2024.acl-long.371/