THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RELAXED SYSTEM LAB

RAG and LLM Agent

COMP6211J

Binhang Yuan

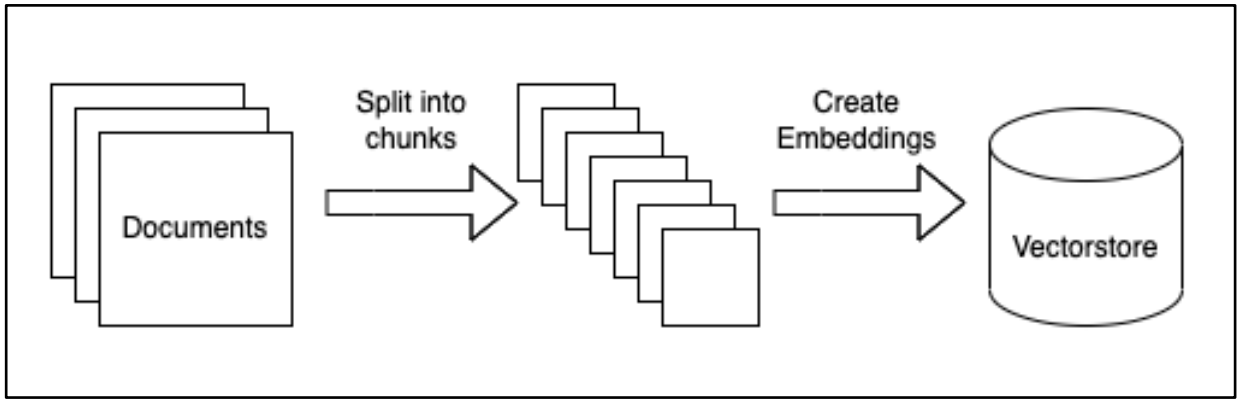
Retrieval Augmented Generation

Retrieval Augmented Generation

RELAXED SYSTEM LAB

- ***Retrieval Augmented Generation (RAG)*** leverages both generative models and retrieval models for knowledge-intensive tasks.
- It improves Generative AI applications by providing up-to-date information and domain-specific data from external data sources during response generation, reducing the risk of ***hallucinations*** and significantly improving performance and accuracy.
  - LLM hallucinations: LLMs produce outputs that are coherent and grammatically correct but factually incorrect or nonsensical.
- Two components to set up RAG:
  - Indexing: ingestion of the data.
  - Retrieval and generation: query the data and augment the generation process with additional information.

RAG Indexing



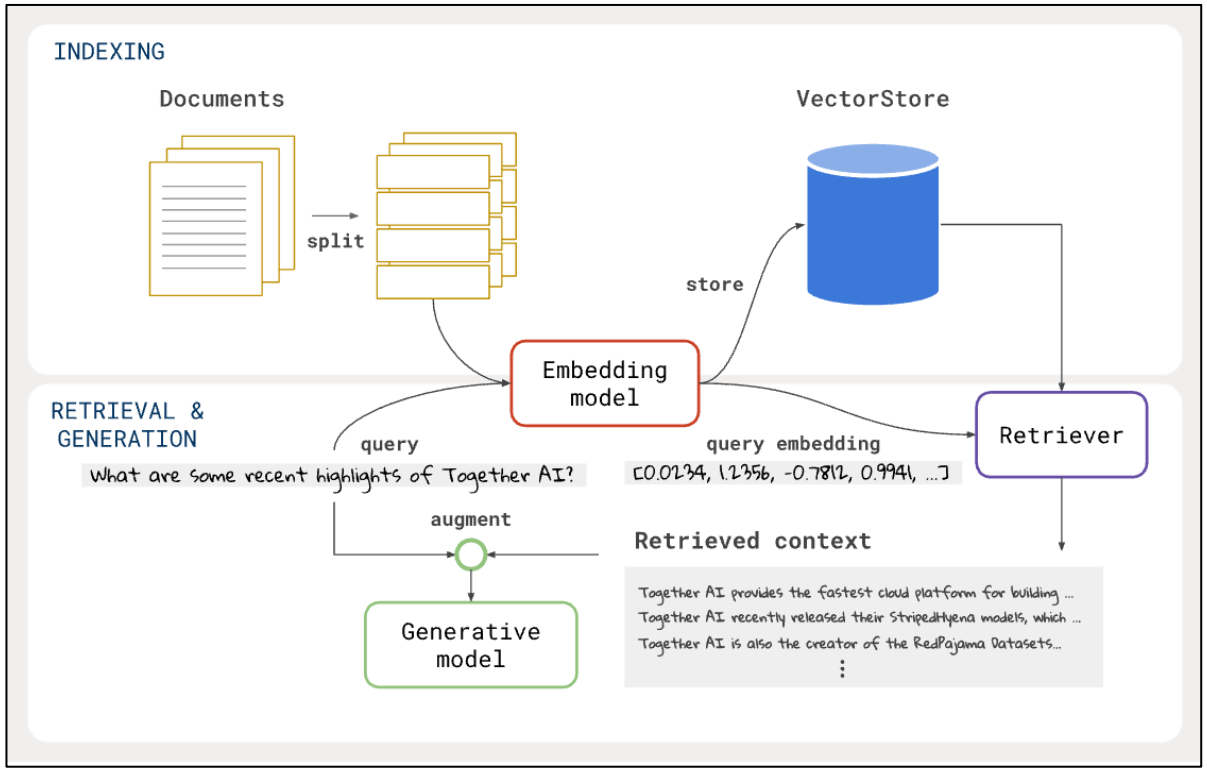
RELAXED SYSTEM LAB

- **Load data sources to text**: load data from arbitrary sources to text in a form that can be used downstream.
- **Chunk text**: chunk the loaded text into smaller chunks. This is necessary because LLMs generally have a limit to a context length limitation, so creating as small chunks of text as possible is necessary.
- **Embed text**: create a numerical embedding for each chunk of text. This is necessary because we only want to select the most relevant chunks of text for a given question, and we will do this by finding the most similar chunks in the embedding space.
- **Load embeddings to vectorDB**: put embeddings and documents into a vectorDB. which helps us find the most similar chunks in the embedding space quickly and efficiently.

RAG Retrieval and Generation

RELAXED SYSTEM LAB

- **Generate the embedding of the input prompt**: this is usually to compute the contextual embedding by the embedding model.
- **Lookup relevant documents**: using the embedding of the input prompt to search the vectorDB created during the indexing phase.
- **Augment the prompt**: Combine the new generation input with the retrieved text into a single prompt.
- **Generate a response**: Given the augmented question, we can use an LLM to generate a response.



RAG Tools

RELAXED
SYSTEM LAB

- LangChain (https://github.com/langchain-ai/langchain): A popular open-source framework that provides building blocks for LLM applications.
- Offer easy integration of LLMs with retrieval components.
- LangChain has ready-made classes for splitting text, creating embeddings, storing vectors, and a RetrievalQA chain that ties a retriever and an LLM together for question-answering.
- Developers can use LangChain to construct custom RAG pipelines without reinventing common steps.

RAG Tools LangChain Example

RELAXED SYSTEM LAB

```python
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.llms import OpenAI
from langchain.chains import RetrievalQA

# 1. Embed and index documents (offline step)
documents = [...]  # list of your text documents
embed_model = OpenAIEmbeddings()  # embedding model (e.g., OpenAI API)
vector_store = FAISS.from_texts(documents, embedding=embed_model)
retriever = vector_store.as_retriever(search_kwargs={"k": 3})

# 2. Set up the RAG chain with an LLM and the retriever
llm = OpenAI(model_name="text-davinci-003", temperature=0)
rag_chain = RetrievalQA(llm=llm, retriever=retriever)

# 3. Ask a question (runtime step)
query = "What is the annual leave policy for employees?"
response = rag_chain.run(query)
print(response)
```

- First, it indexes some documents (e.g. , company HR policies) by embedding them and storing them in a vector index (FAISS).
- At query time, the RetrievalQA chain uses the retriever to find relevant chunks (e.g. , the section of the policy about annual leave) and passes them to the LLM. The LLM then generates an answer like *"Our annual leave policy is ..."* using that retrieved info.

RAG - Embedding Models

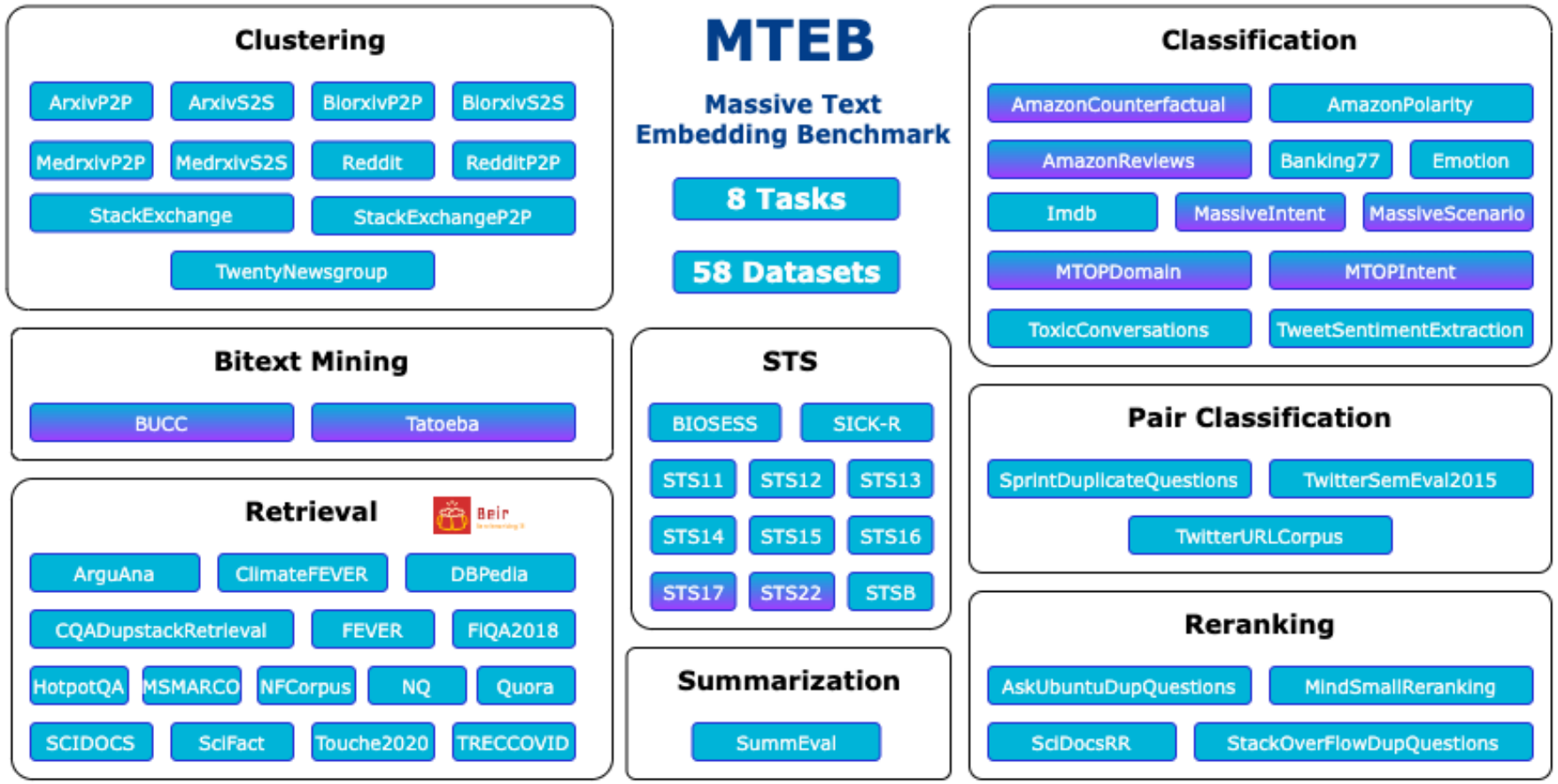
RELAXED SYSTEM LAB

- Text embeddings are vector representations of text.
- Dense Embeddings:
  - Capture semantic relationships between words or sentences.
  - Examples: BERT, Sentence-BERT (SBERT), OpenAI's text-embedding models.
  - Effective for understanding context and meaning beyond exact keyword matches.
- Sparse Embeddings:
  - Focus on exact term frequency and presence.
  - Examples: TF-IDF (Term Frequency-Inverse Document Frequency).
  - Useful for precise keyword-based retrieval.

RAG - Embedding Models

RELAXED SYSTEM LAB

- Evaluating the embedding models:
  - MTEB: Massive Text Embedding Benchmark.



RAG – Text Chunking

RELAXED
SYSTEM LAB

- How to break the original corpus into small pieces?
  - ***Fixed-size (in characters) overlapping sliding window***: divide text into fixed-size chunks based on character count.
  - ***Structure aware splitting (by sentence, paragraph):*** consider the natural structure of text, dividing it based on sentences, paragraphs, sections, or chapters.
  - ***Content-aware splitting (markdown, LaTeX, HTML):*** focus on content type and structure, especially in structured documents like Markdown, LaTeX, or HTML. It ensures content types are not mixed within chunks, maintaining integrity.
  - ***NLP chunking***: track topic changes based on semantic understanding, dividing text into chunks by detecting significant shifts in topics.

Table Augmented Generation

TAG: Table Augmented Generation

RELAXED SYSTEM LAB

- **Goal**: Let users ask arbitrary natural-language questions over data, leveraging both the reasoning of LLMs and the information stored in relational databases.
- Text2SQL converts a natural language query to SQL, but only works for questions expressible in relational algebra.
- RAG augments LLMs with data retrieval, but handles only questions answerable via a few retrieved chunks of text.
- Real user queries often require more: combining up-to-date domain data (i.e., data stored in relational database), world knowledge, and semantic reasoning – e.g., sentiment analysis or trend explanations that pure SQL or simple retrieval can't handle.

Text2SQL is Not Enough: Unifying AI and Databases with TAG

Asim Biswal[1,*] Liana Patel[2,*] Siddharth Jha[1] Amog Kamsetty[1] Shu Liu[1]
Joseph E. Gonzalez[1] Carlos Guestrin[2] Matei Zaharia[1]
[1]UC Berkeley [2]Stanford University

ABSTRACT

AI systems that serve natural language questions over databases promise to unlock tremendous value. Such systems would allow users to leverage the powerful reasoning and knowledge capabilities of language models (LMs) alongside the scalable computational power of data management systems. These combined capabilities would empower users to ask arbitrary natural language questions over custom data sources. However, existing methods and benchmarks insufficiently explore this setting. Text2SQL methods focus solely on natural language questions that can be expressed in relational algebra, representing a small subset of the questions real users wish to ask. Likewise, Retrieval-Augmented Generation (RAG) considers the limited subset of queries that can be answered with point lookups to one or a few data records within the database. We propose Table-Augmented Generation (TAG), a unified and general-purpose paradigm for answering natural language questions over databases. The TAG model represents a wide range of interactions between the LM and database that have been previously unexplored and creates exciting research opportunities for leveraging the world knowledge and reasoning capabilities of LMs over data. We systematically develop benchmarks to study the TAG problem and find that standard methods answer no more than 20% of queries correctly, confirming the need for further research in this area. We release code for the benchmark at https://github.com/TAG-Research/TAG-Bench.

1 INTRODUCTION

Language models promise to revolutionize data management by letting users ask natural language questions over data, which has led to a great deal of research in Text2SQL and Retrieval-Augmented Generation (RAG) methods. In our experience, however (including from internal workloads and customers at Databricks), users' questions often transcend the capabilities of these paradigms, demanding new research investment towards systems that combine the logical reasoning abilities of database systems with the natural language reasoning abilities of modern language models (LMs).

In particular, we find that real business users' questions often require sophisticated combinations of domain knowledge, world knowledge, exact computation, and semantic reasoning. Database systems clearly provide a source of ***domain knowledge*** through the up-to-date data they store, as well as ***exact computation*** at scale (which LMs are bad at).

LMs offer to extend the existing capabilities of databases in two key ways. First, LMs possess ***semantic reasoning*** capabilities over textual data, a core element of many natural language user queries. For example, a Databricks customer survey showed users wish to ask questions like *which customer reviews of product X are positive?*, or *why did my sales drop during this period?*. These questions present complex reasoning-based tasks, such as sentiment analysis over free-text fields or summarization of trends. LMs are well-suited to these tasks, which cannot be modeled by the exact computation or relational primitives in traditional database systems.

Secondly, the LM, using knowledge learned during model training and stored implicitly by the model's weights, can powerfully augment the user's data with ***world knowledge*** that is not captured explicitly by the database's table schema. As an example, a Databricks internal AI user asked *what are the QoQ trends for the "retail" vertical?* over a table containing attributes for account names, products and revenue. To answer this query the system must understand how the business defines *QoQ* (e.g., the quarter over quarter trends from the last quarter to the current quarter or this quarter last year to this quarter this year), as well as which companies are considered to be in the *retail vertical*. This task is well-suited to leverage the knowledge held by a pre-trained or fine-tuned LM.

Systems that efficiently leverage databases and LMs together to serve natural language queries, in their full generality, hold potential to transform the way users understand their data. Unfortunately, these questions cannot be answered today by common methods, such as Text2SQL and RAG. While Text2SQL methods [26, 28, 31, 32] are suitable for the subset of natural language queries that have direct relational equivalents, they cannot handle the vast array of user queries that require semantic reasoning or world knowledge. For instance, the previous user query asking *which customer reviews are positive* may require logical row-wise LM reasoning over reviews to classify each as positive or negative. Similarly the question which asks *why sales dropped* entails a reasoning question that must aggregate information across many table entries.

On the other hand, the RAG model is limited to simple relevance-based point lookups to a few data records, followed by a single LM invocation. This model serves only the subset of queries answerable by point lookups and also fails to leverage the richer query execution capabilities of many database systems, which leaves computational tasks (e.g., counting, math and filtering) to a single invocation of the error-prone LM. In addition to being error prone and inefficient at computational tasks, LMs have also been shown to perform poorly on long-context prompts limiting their ability to reason about data at scale in the generation phase of RAG.

We instead propose ***table-augmented generation (TAG)*** as a unified paradigm for systems that answer natural language questions over databases. Specifically, TAG defines three key steps, as shown in Figure 1. First, the query synthesis step syn translates the user's arbitrary natural language request R to an executable database query Q. Then, the query execution step exec executes Q on the database system to efficiently compute the relevant data T. Lastly, the answer generation step gen utilizes R and T, where the LM is orchestrated, possibly in iterative or recursive patterns over the data, to generate the final natural language answer A. The TAG model is simple, but powerful: it is defined by the following three

*Both authors contributed equally to this research.

Text2SQL Example

RELAXED
SYSTEM LAB

🧾 Natural Language Question

"What are the names and email addresses of customers who placed orders in the last 30 days?"

🗂️ Database Schema

Tables:

- **Customers**
  - `customer_id` (INTEGER)
  - `name` (TEXT)
  - `email` (TEXT)
- **Orders**
  - `order_id` (INTEGER)
  - `customer_id` (INTEGER)
  - `order_date` (DATE)

🤖 LLM-Generated SQL Query

```sql
SELECT c.name, c.email
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.order_date >= CURRENT_DATE - INTERVAL '30 days';
```

Limitation of Text2SQL

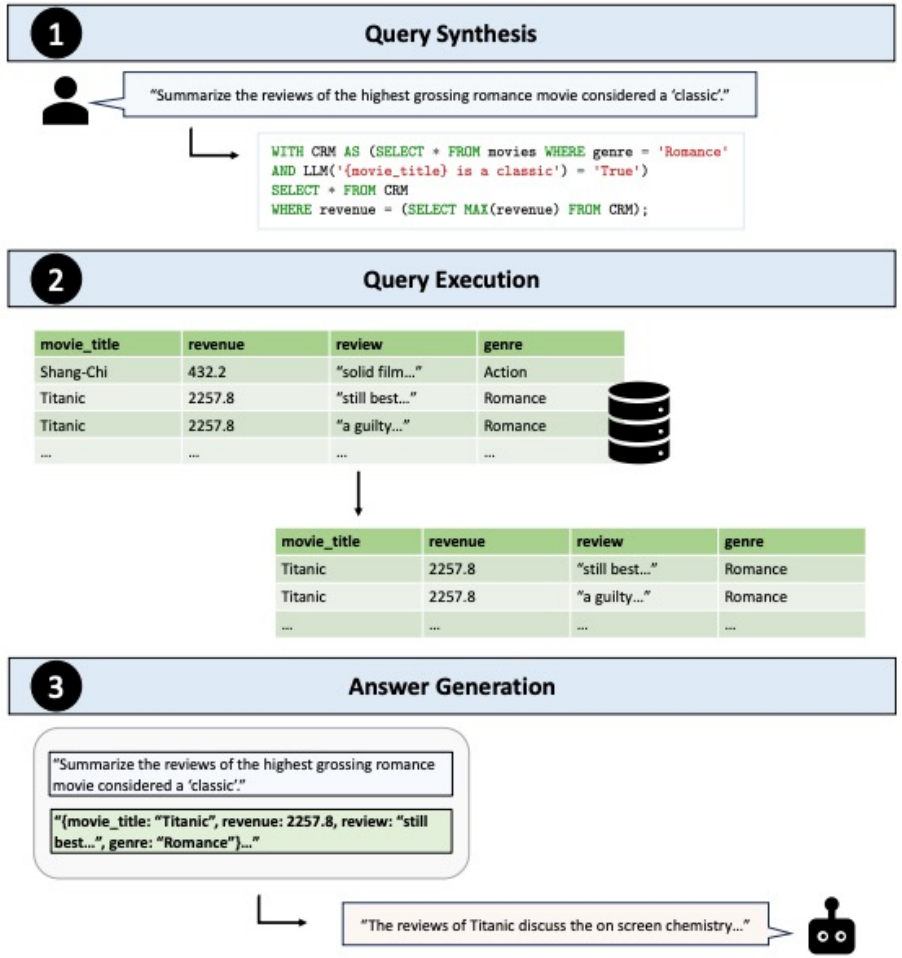
RELAXED SYSTEM LAB

- **Structured queries only**: the question must be fully translated into SQL (i.e., has a direct relational form); it cannot perform tasks beyond SQL's capabilities.
- **No semantic reasoning or external knowledge**: it cannot handle queries requiring understanding free text or using facts not in the database. For example:
  - "Which customer reviews of product X are positive?" – Requires classifying review sentiment (an NL task), which SQL alone cannot do.
  - "Why did my sales drop during this period?" – Involves reasoning across many data points and explaining a trend beyond a single SQL query's scope.
- **Covers a small subset of needs**: In practice, Text2SQL can answer only a limited fraction of realistic data questions.

TAG Pipeline: Three Key Steps

RELAXED SYSTEM LAB

- **Query Synthesis**: The LLM parses the natural language request and generates a database query (e.g. in SQL) to retrieve the needed information.
- **Query Execution:** The generated query is executed on the database, which efficiently processes it over potentially large data.
- **Answer Generation**: Finally, the LLM generates a natural-language answer using the results from the database. It takes the retrieved data (often formatted or summarized into text) and composes the answer to the user's question.



Real-World TAG Scenarios

RELAXED
SYSTEM LAB

- **Sentiment Analysis & Feedback Mining**:
  - TAG enables analytics on textual data stored in tables.
  - For example: "Among the negative reviews of product X, what are the most frequently criticized features?"
  - A TAG system can use the LLM to classify each review as positive/negative during Query Synthesis and filter the table, then aggregate common complaints and have the LLM summarize them.
  - This provides insights that neither SQL nor simple lookup could produce alone.

Real-World TAG Scenarios

RELAXED
SYSTEM LAB

- **Business Queries with World Knowledge**:
  - TAG can handle enterprise questions that involve jargon or background knowledge not explicitly in the database.
  - E.g.: "What are the QoQ trends for the 'retail' vertical in our sales data?"
  - The term "retail" might not be a column in the data, but the LLM can infer which entries (companies) are in the retail sector, and the database can compute quarter-over-quarter trends for those, which the LLM then narrates.

Real-World TAG Scenarios

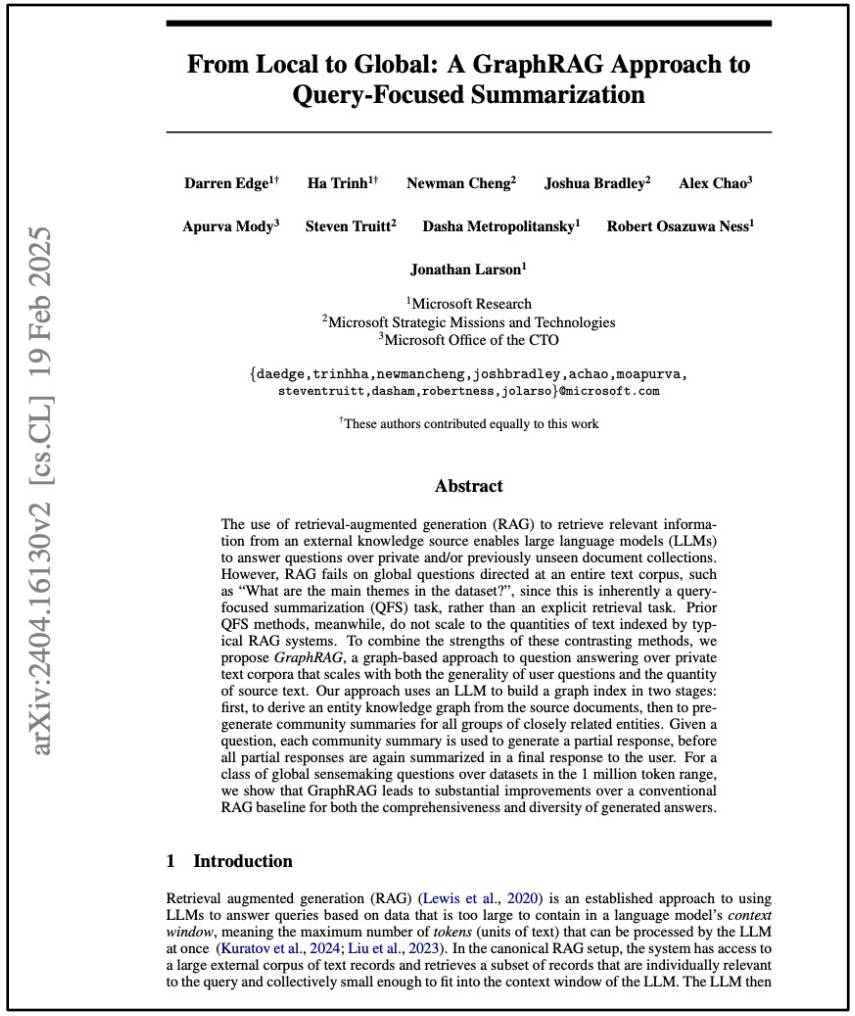
RELAXED
SYSTEM LAB

- **Data Summarization at Scale**:
  - TAG shines at questions requiring combining many data points.
  - For instance: "Summarize all races held at Sepang International Circuit."
  - A TAG system can execute an SQL query to pull all records of races at that circuit and then use the LLM to generate a concise summary of the results (e.g., listing years, winners, and notable facts).
  - Prior approaches would either miss many entries or struggle to compile an answer from dozens of rows.

GraphRAG

GraphRAG Motivation

- Challenges with traditional RAG:
  - Ineffective for "sensemaking" queries like "What are the main themes in the dataset?"
  - Retrieval based on local relevance, missing broader connections.
- Need for structured context:
  - Incorporating relationships between entities can provide deeper insights.
  - Graph structures naturally represent interconnected information.
- Core idea:
  - GraphRAG integrates graph-based representations into the RAG framework.
  - Constructs a knowledge graph from the source corpus to guide retrieval and generation.



From Local to Global: A GraphRAG Approach to Query-Focused Summarization

Darren Edge[1†] Ha Trinh[1†] Newman Cheng[2] Joshua Bradley[2] Alex Chao[3]

Apurva Mody[3] Steven Truitt[2] Dasha Metropolitansky[1] Robert Osazuwa Ness[1]

Jonathan Larson[1]

[1]Microsoft Research
[2]Microsoft Strategic Missions and Technologies
[3]Microsoft Office of the CTO

{daedge,trinhha,newmancheng,joshbradley,achao,moapurva,
steventruitt,dasham,robertness,jolarso}@microsoft.com

[†]These authors contributed equally to this work

Abstract

The use of retrieval-augmented generation (RAG) to retrieve relevant information from an external knowledge source enables large language models (LLMs) to answer questions over private and/or previously unseen document collections. However, RAG fails on global questions directed at an entire text corpus, such as "What are the main themes in the dataset?", since this is inherently a query-focused summarization (QFS) task, rather than an explicit retrieval task. Prior QFS methods, meanwhile, do not scale to the quantities of text indexed by typical RAG systems. To combine the strengths of these contrasting methods, we propose *GraphRAG*, a graph-based approach to question answering over private text corpora that scales with both the generality of user questions and the quantity of source text. Our approach uses an LLM to build a graph index in two stages: first, to derive an entity knowledge graph from the source documents, then to pre-generate community summaries for all groups of closely related entities. Given a question, each community summary is used to generate a partial response, before all partial responses are again summarized in a final response to the user. For a class of global sensemaking questions over datasets in the 1 million token range, we show that GraphRAG leads to substantial improvements over a conventional RAG baseline for both the comprehensiveness and diversity of generated answers.

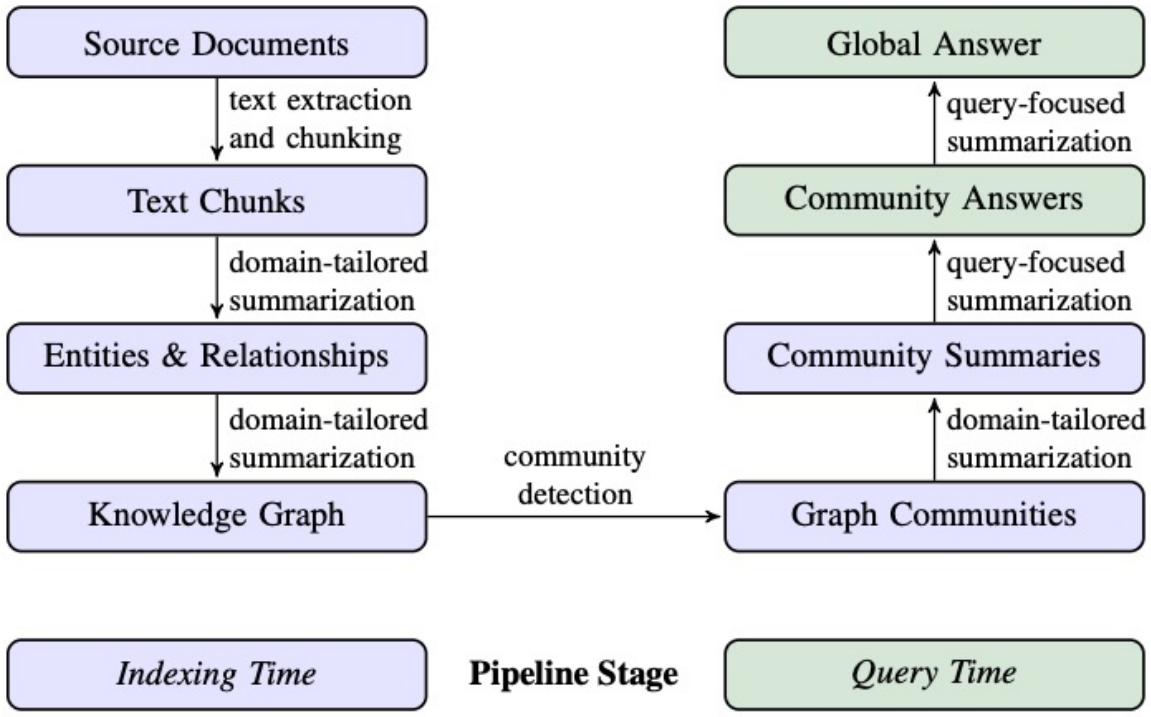
1 Introduction

Retrieval augmented generation (RAG) (Lewis et al., 2020) is an established approach to using LLMs to answer queries based on data that is too large to contain in a language model's *context window*, meaning the maximum number of *tokens* (units of text) that can be processed by the LLM at once (Kuratov et al., 2024; Liu et al., 2023). In the canonical RAG setup, the system has access to a large external corpus of text records and retrieves a subset of records that are individually relevant to the query and collectively small enough to fit into the context window of the LLM. The LLM then

GraphRAG Architecture Overview

RELAXED SYSTEM LAB

- ***Indexing time***:
  - **Source documents -> text chunks**: the documents in the corpus are split into text chunks.
  - **Entity and relation extraction**: using LLMs to identify entities and their relationships.
  - **Knowledge graph construction**: forming a knowledge graph from extracted entities and relations.
  - **Graph community detection**: identifying clusters within the graph.
  - **Community summarization**: Generating summaries for each community.
- ***Query time***:
  - **Query processing**: Matching user queries to relevant community summaries.
  - **Answer generation**: LLM generates responses based on selected summaries.



Case Study - Entity and Relation Extraction

RELAXED SYSTEM LAB

Scenario: Analyzing a corpus of pharmaceutical research papers to understand drug interactions and side effects.

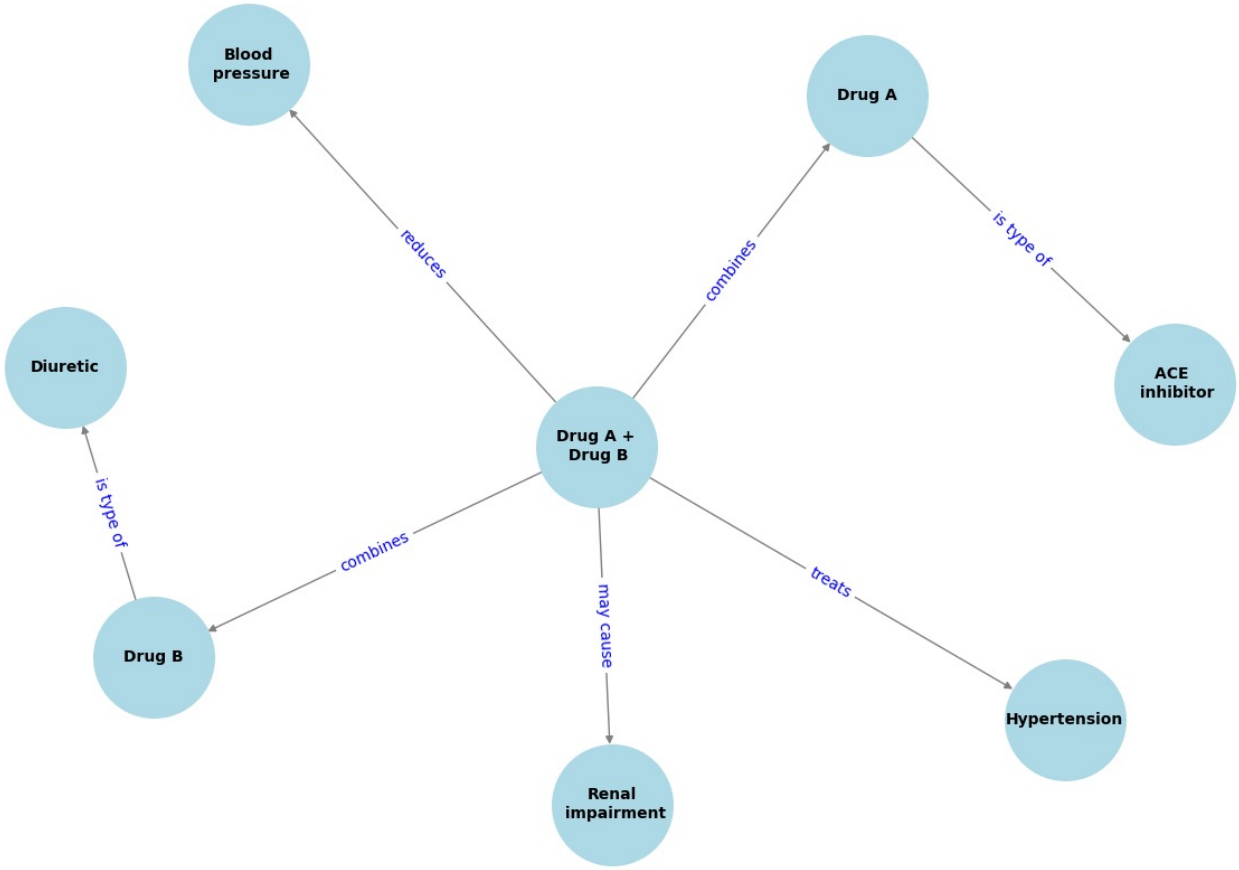
Input Text: "A recent study indicates that combining Drug A (an ACE inhibitor) with Drug B (a diuretic) significantly reduces blood pressure in patients with hypertension. However, this combination may increase the risk of renal impairment."

- **Extracted Entities**:
  - Drug A
  - ACE inhibitor
  - Drug B
  - Diuretic
  - Blood pressure
  - Hypertension
  - Medium
  - Renal impairment
- **Extracted Relations**:
  - (Drug A, is a type of, ACE inhibitor)
  - (Drug B, is a type of, Diuretic)
  - (Drug A + Drug B, reduces, Blood pressure)
  - (Drug A + Drug B, treats, Hypertension)
  - (Drug A + Drug B, may cause, Renal impairment

Case Study - Knowledge Graph Construction

RELAXED SYSTEM LAB

- **Extracted Entities**:
  - Drug A
  - ACE inhibitor
  - Drug B
  - Diuretic
  - Blood pressure
  - Hypertension
  - Medium
  - Renal impairment
- **Extracted Relations**:
  - (Drug A, is a type of, ACE inhibitor)
  - (Drug B, is a type of, Diuretic)
  - (Drug A + Drug B, reduces, Blood pressure)
  - (Drug A + Drug B, treats, Hypertension)
  - (Drug A + Drug B, may cause, Renal impairment
- The combination of Drug A and Drug B is represented as a composite node to capture their combined effects.



Case Study - Community Detection and Summarization

RELAXED SYSTEM LAB

Community Detection:

- **Community 1**: Antihypertensive Agents
  - Drug A
  - ACE inhibitor
  - Drug B
  - Diuretic
  - Blood pressure
  - Hypertension
- **Community 2**: Adverse Effects
  - Drug A + Drug B
  - Renal impairment

Community Summaries:

- **Antihypertensive Agents**:
  - "Drug A, an ACE inhibitor, and Drug B, a diuretic, are commonly used in combination to effectively reduce blood pressure in hypertensive patients."
- **Adverse Effects**:
  - "While the combination of Drug A and Drug B is effective in managing hypertension, it may increase the risk of renal impairment."

Case Study - Query Processing and Answer Generation

RELAXED SYSTEM LAB

User Query: "What are the benefits and risks of combining ACE inhibitors with diuretics in hypertension treatment?"

Processing Steps:

- **Query Understanding**: Identify key entities: ACE inhibitors, diuretics, hypertension, benefits, risks.
- **Community Matching**: Match query to relevant communities: Antihypertensive Agents and Adverse Effects.
- **Information Retrieval**: Retrieve summaries and relations from matched communities.

Answer Generation: "Combining ACE inhibitors with diuretics is effective in lowering blood pressure among hypertensive patients. However, this combination may increase the risk of renal impairment, necessitating careful monitoring."

Comparison with Standard RAG

RELAXED SYSTEM LAB

- Standard RAG:
  - Relies on vector similarity for retrieval.
  - Limited in capturing structural relationships.
- GraphRAG:
  - Utilizes graph structures to represent knowledge.
  - Enhances retrieval by considering entity relationships and community contexts.
- Example:
  - Query: "What are the main themes in the dataset?"
  - *Standard RAG Response*: May retrieve unrelated documents based on keyword similarity.
  - *GraphRAG Response*: Provides a summary of interconnected themes identified through community detection.

RELAXED
SYSTEM LAB

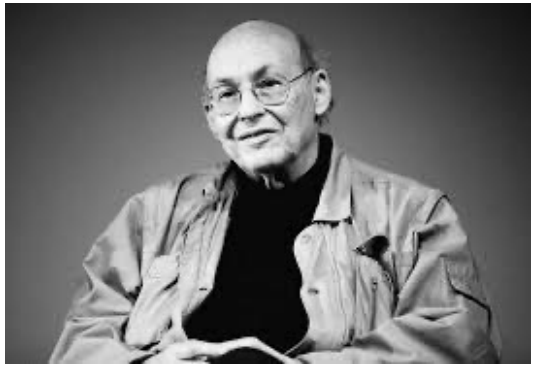
AI Agent

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

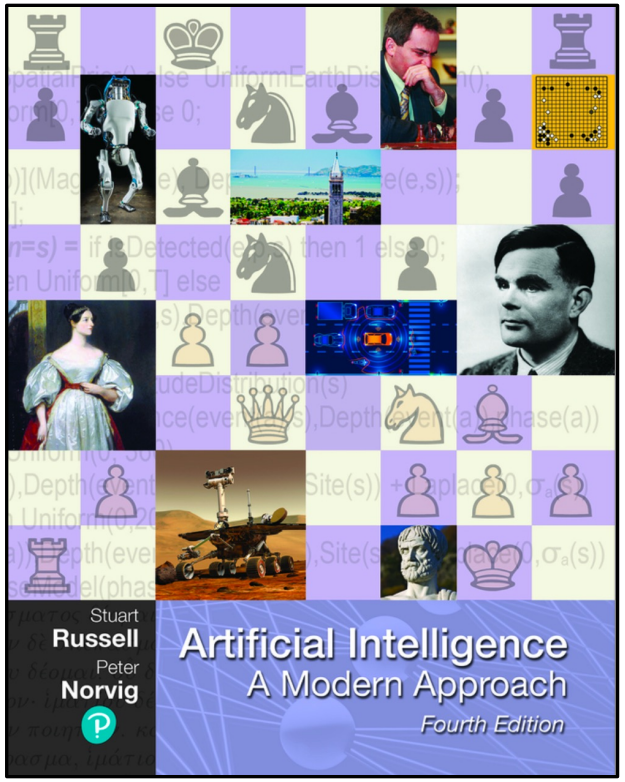
- *"AI is the science of making machines do things that would require intelligence if done by men."* – Marvin Minsky



A Little Background about AI

RELAXED
SYSTEM LAB

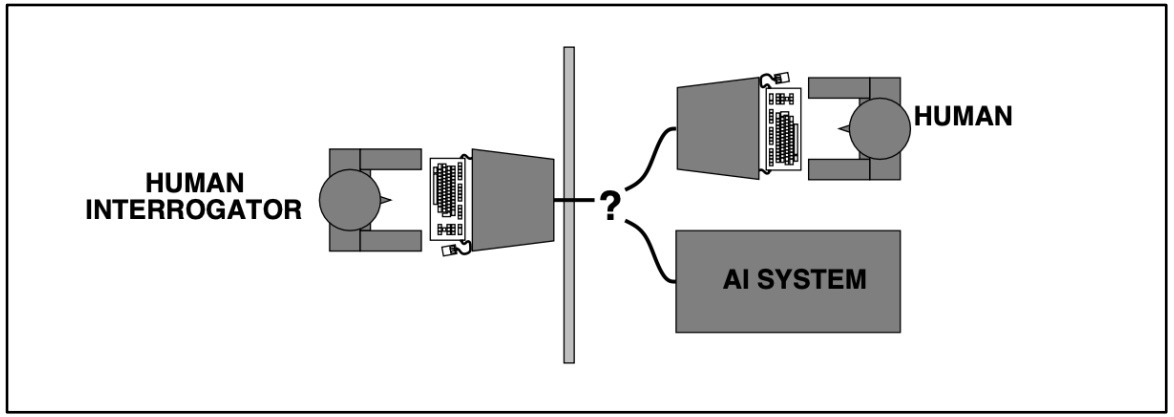
- What is AI?
- Four possible combinations from two dimensions:
  - Human vs. Rational
  - Thought vs. Behaviour
- The view of AI falls into four categories:
  - Acting humanly
  - Thinking humanly
  - Thinking rationally
  - Acting rationally



What is AI? Acting Humanly

RELAXED SYSTEM LAB

- ***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

RELAXED
SYSTEM LAB

- ***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

RELAXED
SYSTEM LAB

- ***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

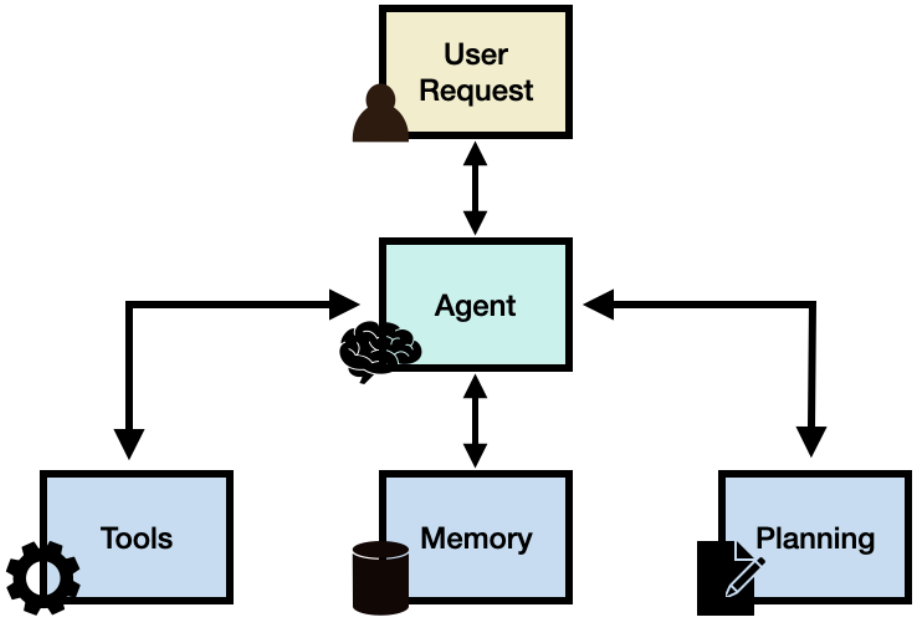
- **Single-Agent Framework**:
  - A single-agent system utilizes one AI entity to manage all tasks, encompassing planning, execution, and decision-making.
- **Multi-Agent Framework**:
  - A multi-agent system comprises multiple AI agents, each specialized in distinct tasks, collaborating to achieve a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

LLM Agent

LLM Agent Framework

RELAXED SYSTEM LAB

- 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

RELAXED
SYSTEM LAB

- 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

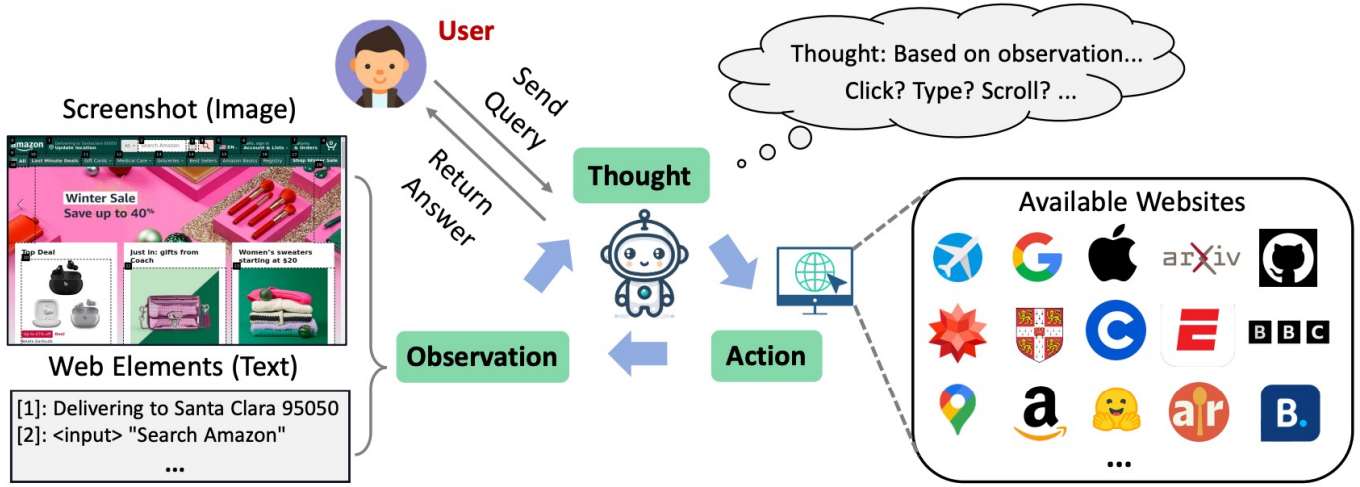
RELAXED SYSTEM LAB

- 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.

LLM Agent Case Study — WebVoyager

WebVoyager

RELAXED SYSTEM LAB



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

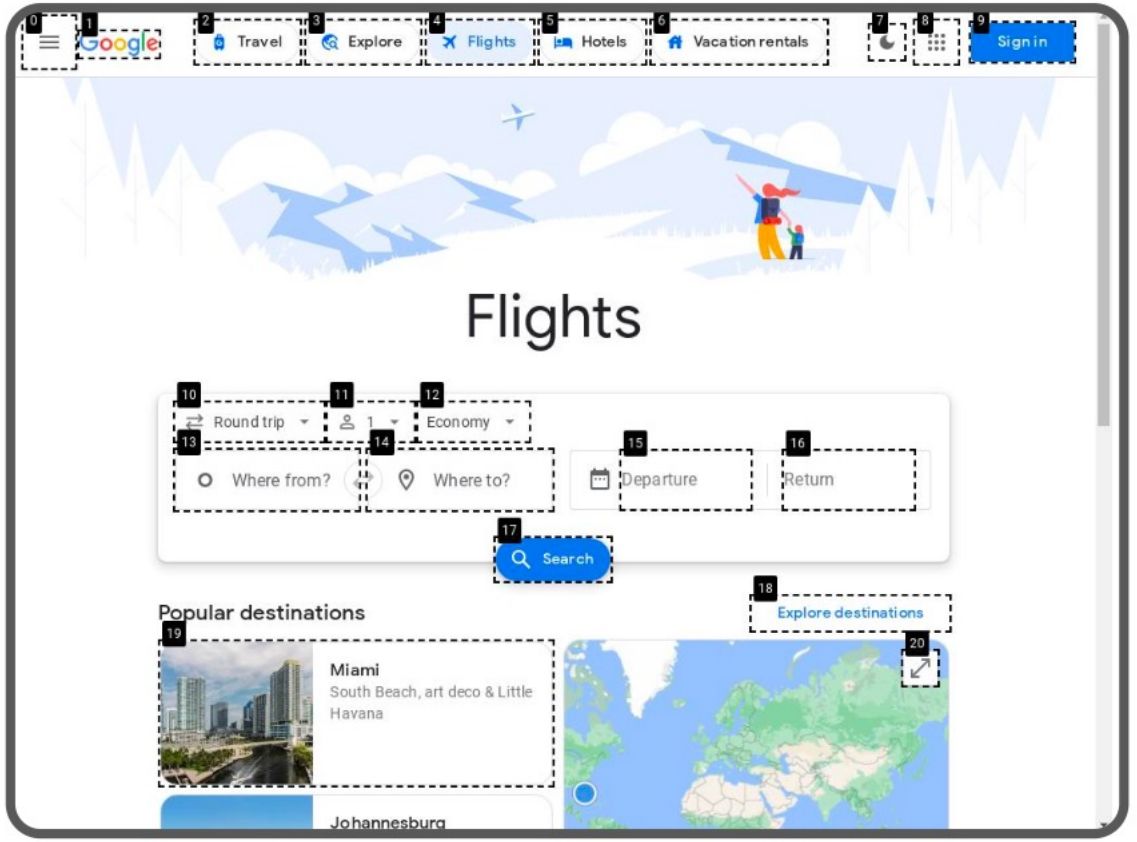
Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6864–6890
August 11-16, 2024 ©2024 Association for Computational Linguistics

WebVoyager

RELAXED
SYSTEM LAB

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.

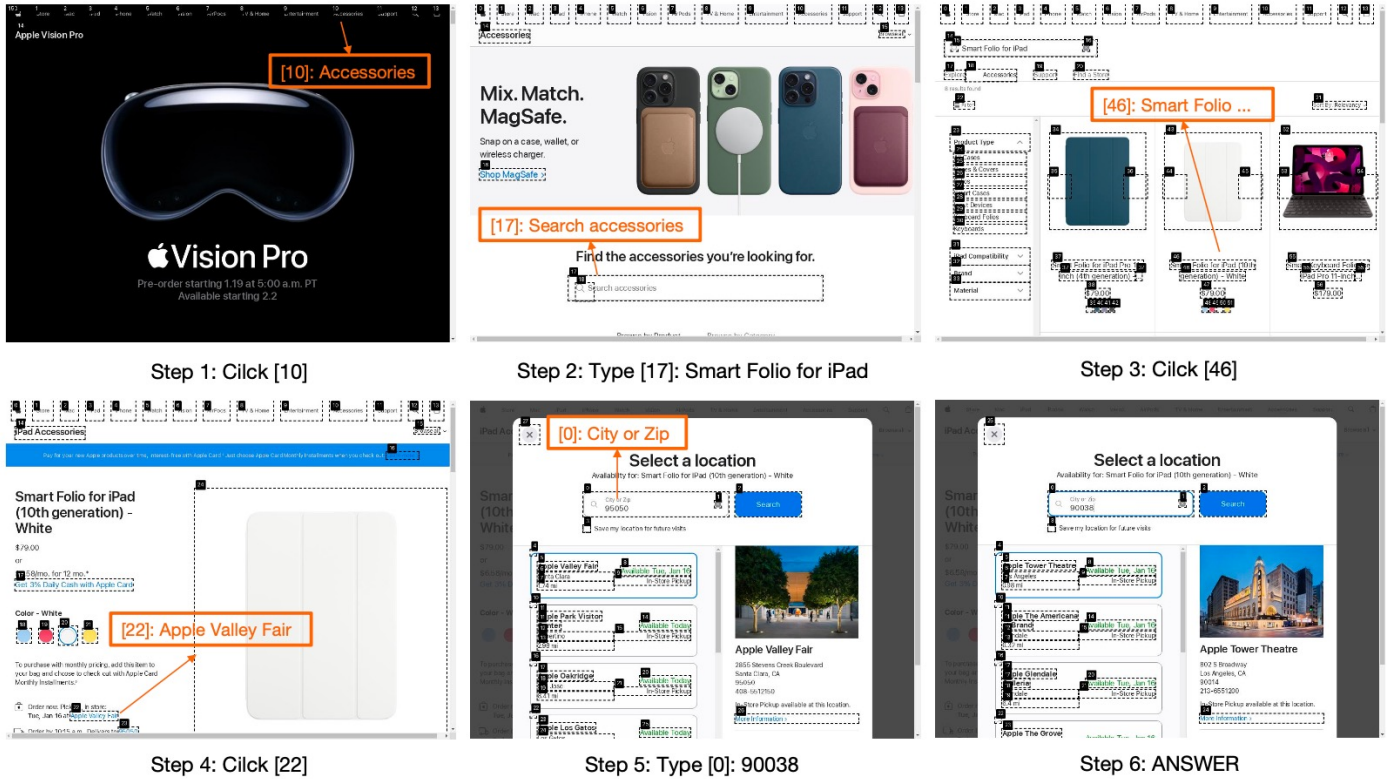


WebVoyager

RELAXED SYSTEM LAB

Action Space:

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



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.

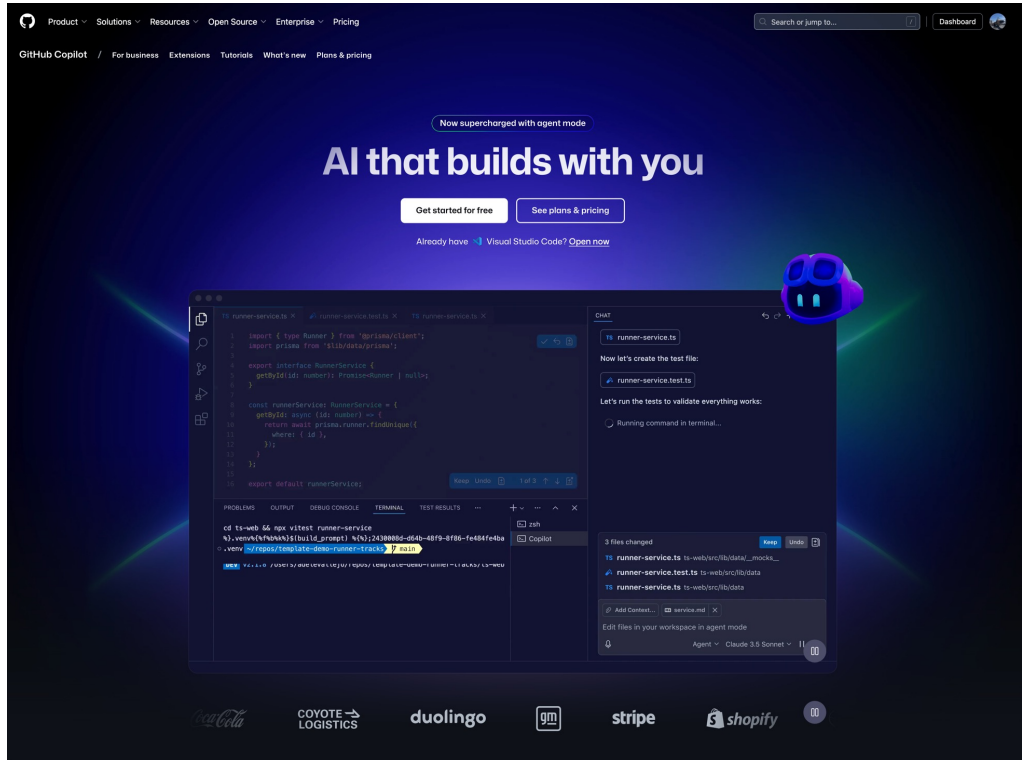
The process continues until the agent decides to stop.

LLM Agent Case Study — GitHub Copilot

GitHub Copilot: Coding Assistant

RELAXED SYSTEM LAB

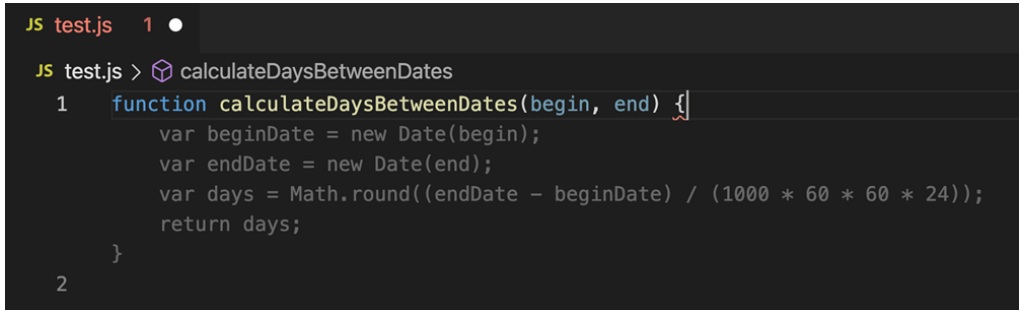
- Copilot is an AI pair programmer tool in Visual Studio Code.
- Provide code suggestions as you type in the editor:
  - **AI Code completions**: start typing in the editor, and Copilot provides code suggestions based on your existing code that matches your coding style.
- Use natural language chat to 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.



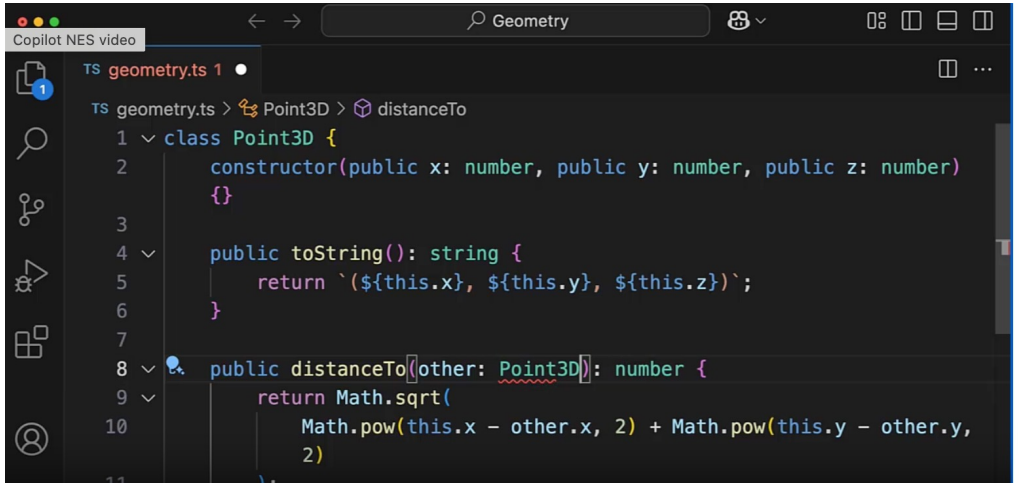
GitHub Copilot: AI Code Completions

RELAXED SYSTEM LAB

- Two kinds of code completions:
- **Code completions**: Start typing in the editor, and Copilot provides code suggestions that match your coding style and take your existing code into account.
- **Next Edit Suggestions**: 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.



Code Completions

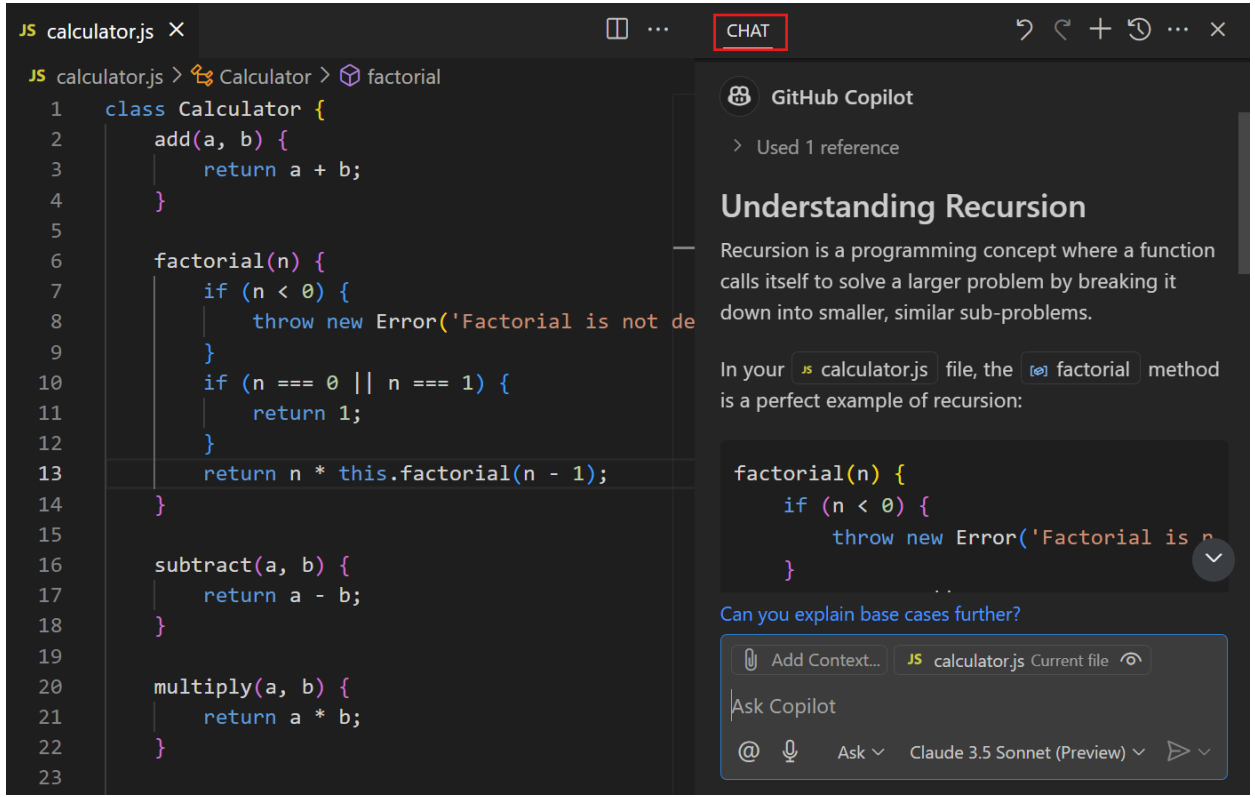


Next Edit Suggestions

GitHub Copilot: Natural Language Chat

RELAXED SYSTEM LAB

- 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?"



Natural Language Chat

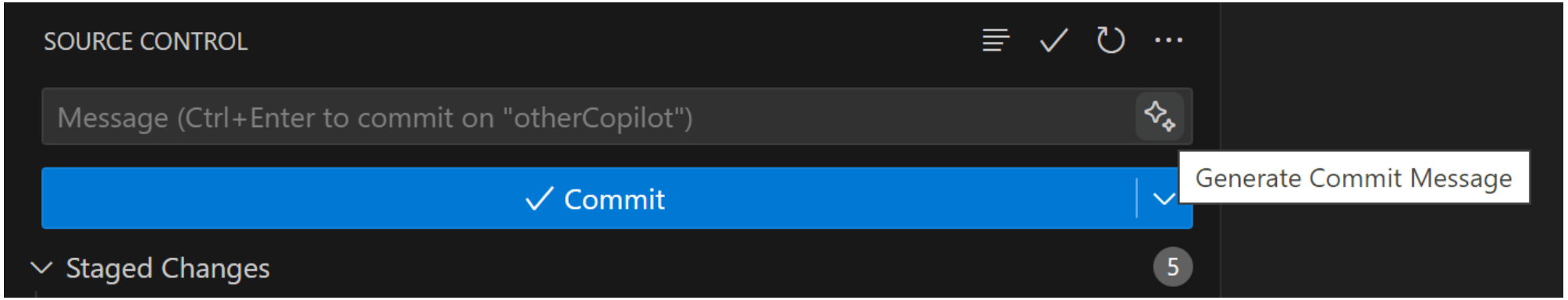
GitHub Copilot: Smart Actions

RELAXED SYSTEM LAB

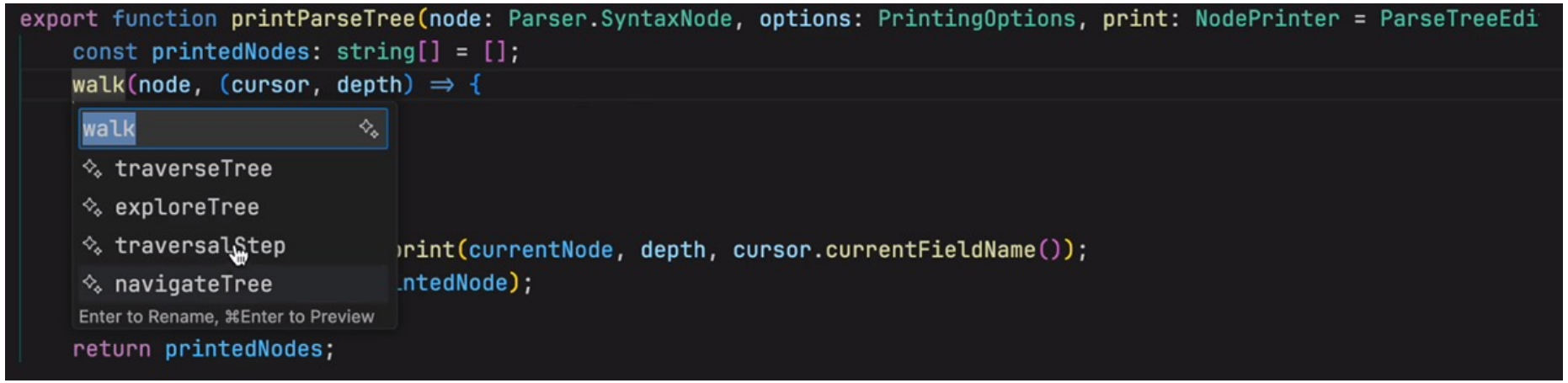
- For several common scenarios, you can use smart actions to get help from Copilot without having to write a prompt:
  - Generate a commit message and PR information;
  - Rename symbols;
  - Generate documentation;
  - Generate tests;
  - Explain code;
  - Fix coding errors;
  - Fix testing errors;
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

GitHub Copilot: Smart Actions

RELAXED SYSTEM LAB



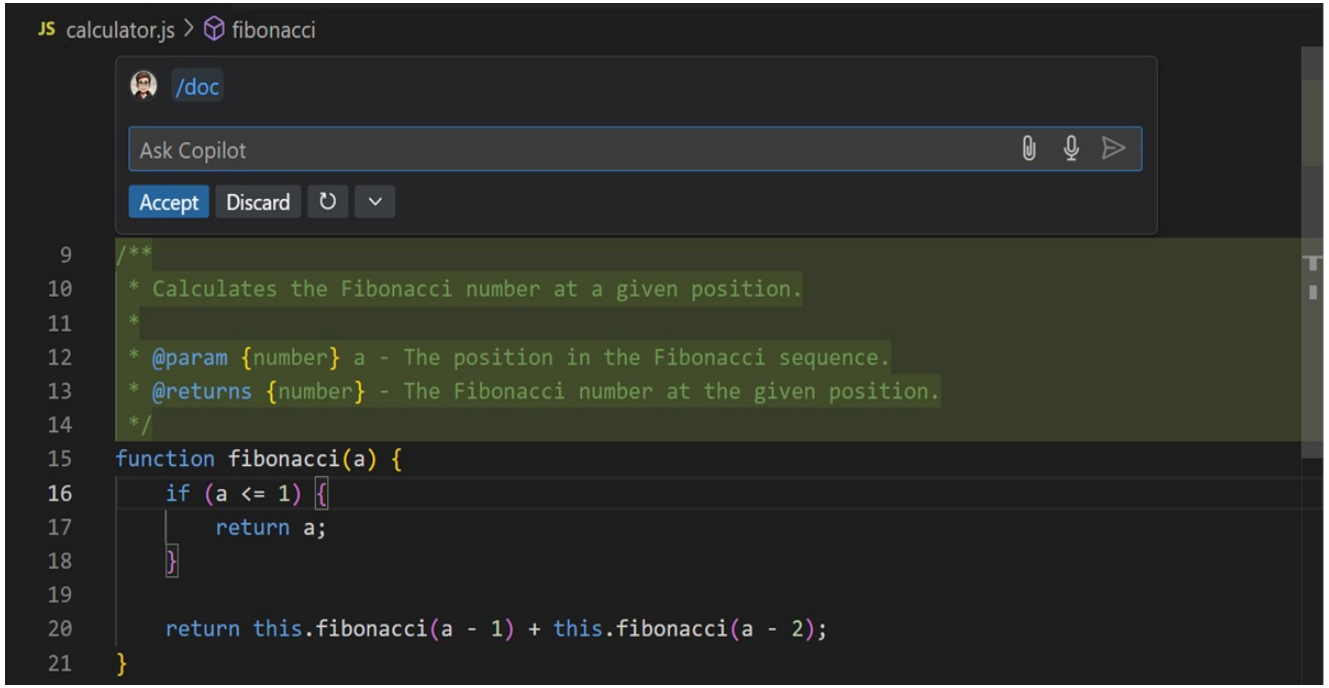
Generate a commit message and PR information



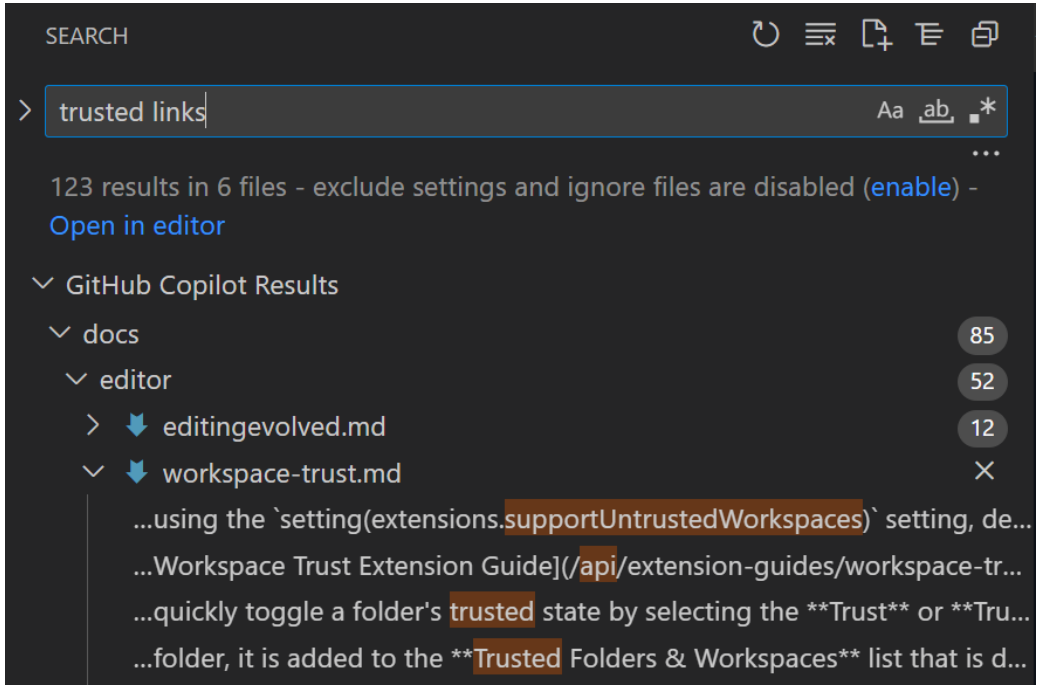
Rename symbols

GitHub Copilot: Smart Actions

RELAXED SYSTEM LAB



Generate documentation



Semantic search results

RELAXED
SYSTEM LAB

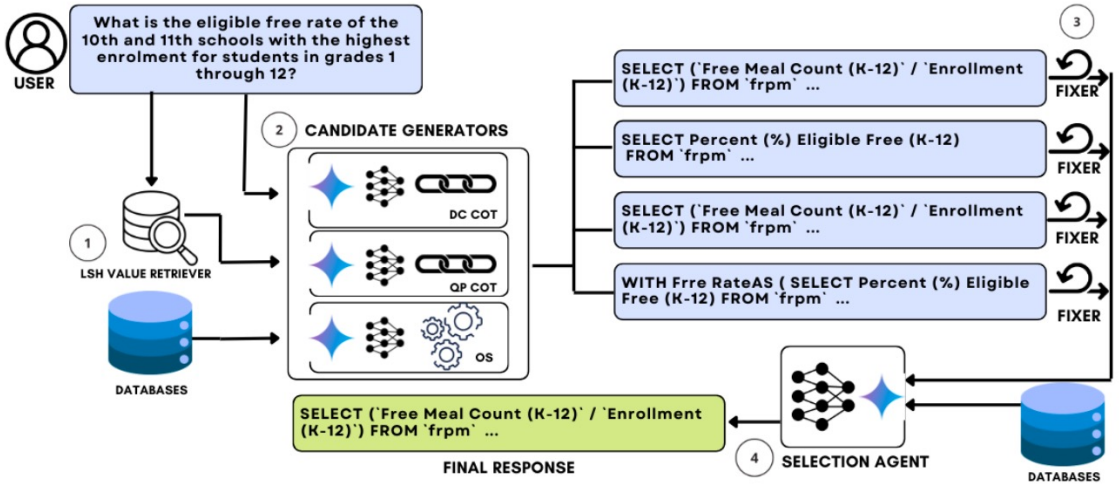
LLM Agent Case Study — CHASE-SQL

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.

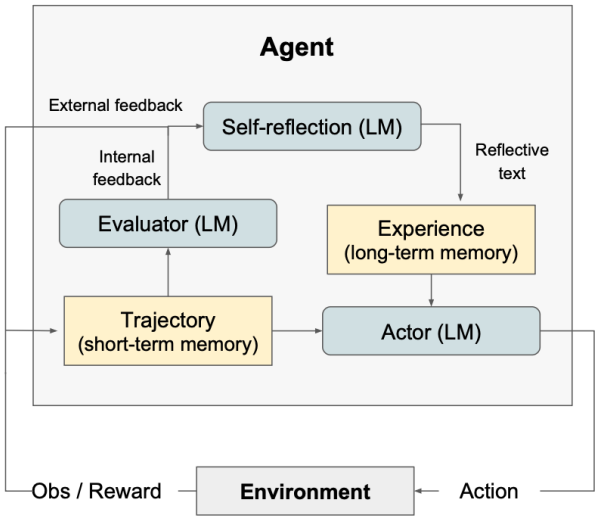


- 3. **Query fixer**: ***"self-reflects"*** on the previously generated query, using feedback such as syntax error details or empty result sets to guide the correction process
- 4. **Selection agent**: Choose the best SQL query from the set of viable candidates.

Self-Reflection in LLM Agent

RELAXED SYSTEM LAB

- 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 π_θ
Evaluate τ_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 $\tau_t = [a_0, o_0, \ldots a_i, o_i]$ using π_θ
  Evaluate τ_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://www.together.ai/blog/rag-tutorial-langchain
- https://huggingface.co/blog/mteb
- https://arxiv.org/pdf/2408.14717
- https://arxiv.org/abs/2404.16130
- 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/