


Spring 2024

LociGraph: AI Agent Framework for Browser-Based Knowledge Graph Construction

Nathan Cho
Bard College

Follow this and additional works at: https://digitalcommons.bard.edu/senproj_s2024

 Part of the [Artificial Intelligence and Robotics Commons](#), [Databases and Information Systems Commons](#), and the [Software Engineering Commons](#)



This work is licensed under a [Creative Commons Attribution 4.0 License](#).

Recommended Citation

Cho, Nathan, "LociGraph: AI Agent Framework for Browser-Based Knowledge Graph Construction" (2024). *Senior Projects Spring 2024*. 46.

https://digitalcommons.bard.edu/senproj_s2024/46

This Open Access is brought to you for free and open access by the Bard Undergraduate Senior Projects at Bard Digital Commons. It has been accepted for inclusion in Senior Projects Spring 2024 by an authorized administrator of Bard Digital Commons. For more information, please contact digitalcommons@bard.edu.



**LociGraph: AI Agent Framework for
Browser-Based Knowledge Graph Construction**

Senior Project Submitted to
The Division of Science, Math, and Computing
of Bard College

by
Nathan Cho

Annandale-on-Hudson, New York
May 2024

Abstract

This work presents **LociGraph**, an artificial intelligence agent that can autonomously search information on the non-public web, such as email inboxes, online communities, social media, or web applications not searchable on a public search engine. With a given query, the agent will browse the web using the keyboard and mouse to find a webpage containing the relevant information and extract the information in a structured format. For example, if the agent is given the query [Alex, studied at, ?] on your email inbox, the agent will start by typing “Alex” into the search bar, click on email related to Alex, read the content “Alex went to Bard College” and return [Alex, studied at, Bard College]. The framework consists of two parts: an *agent pipeline*, where a group of agents analyze the webpage content and suggest the next action, and a *browser extension*, where the user can enter the query and execute the suggested action. Preliminary evaluations show that Large Language Model (LLM) agents can navigate and extract information from real-world websites, but struggle to extract information from indirectly relevant content. All code, benchmarks, and results are available at <https://github.com/ntcho/LociGraph>.

Acknowledgments

I would like to thank my advisor, Rose Sloan, for her guidance and support throughout my senior project and academic life at Bard. Additionally, I am extremely grateful for Bard College supporting my academic endeavor with a generous scholarship, allowing me to reach new places that I never thought I had a chance.

To my friends, most of all the Bard Pickup and the Bard Computer Science Club, thank you for all the time we spent together – I will cherish those moments whenever I remember my college years.

To Anna, thank you for your love, for being my mentor and friend, for the best snacks, and for inspiring me to be the best version of myself.

To my family, thank you for always believing in me, supporting my endeavors, and sharing love.

To Debbie, none of this would have been possible without your unconditional love.

*I love you, LORD; you are my strength. The LORD is my rock, my fortress, and my savior;
my God is my rock, in whom I find protection. He is my shield, the power that saves me,
and my place of safety.*

Psalms 18:1-2

Table of Contents

1 Introduction.....	1
2 Related Work.....	4
2.1 Knowledge Graph Construction.....	4
2.2 Autonomous Web Navigation.....	6
2.3 Webpage Representation.....	8
3 LociGraph Framework.....	9
3.1 Agent Pipeline.....	11
3.2 Browser Extension.....	12
4 Implementation.....	13
4.1 Agent Pipeline.....	13
4.1.1 Webpage Parsing.....	14
4.1.2 Element Ranking.....	23
4.1.3 Relation Extraction.....	25
4.1.4 Result Evaluation.....	28
4.1.5 Action Prediction.....	30
4.2 Browser Extension.....	32
4.2.1 User Interface.....	32
4.2.2 Webpage Encoding.....	34
4.2.3 Action Execution.....	34
4.3.4 Graph Visualization.....	35
5 Experiments.....	37
5.1 Quantitative Assessment: Wikipedia.....	37
5.2 Qualitative Assessment: Gmail.....	38
6 Discussion.....	38
6.1 Experimental Findings.....	38
6.2 Limitations.....	39
6.3 Future Work.....	39
7 Conclusion.....	41
References.....	42
Appendix.....	46
A Implementation Details.....	46
B Experiment Details.....	51
C Source Code.....	65

1 Introduction

Each person interacts with hundreds of webpages every day. Between those hundreds of webpages, information is stored in hundreds of different unstructured formats such as email, messages, posts, and profiles in various web applications. This lack of structure and separation of data makes it very difficult to manage all the information one has access to in a centralized and organized way. For example, imagine you want to know where your colleague went for college. You vaguely remember it was mentioned somewhere in email or social media, but this information is not publicly available online. In order to find this piece of information, you have to open each webpage and manually search for your colleague.

Recent developments in generative artificial intelligence technology, especially large language models (LLMs) such as ChatGPT [1], have shown that AI models are capable of being an agent that takes actions to achieve an objective. Following this development, many researchers developed AI agents that can autonomously navigate through webpages to achieve a wide range of objectives. [2] This research focuses on building agents for general-purpose tasks, such as searching for information, shopping items, and interacting with web applications, where the final output of the agent is another webpage or a conversational response. For example, using the objective “Find where Alex went to college” on general-purpose agents will result in a webpage that contains this information or a response such as “Alex went to Bard College.” Although this output is useful, the information is contained in an unstructured format, leaving another step of data cleaning to make the information useful at scale. This limitation raises an important research question: *how can we build an agent that can extract structured data from non-public webpages?*

Motivated by this question, this work defines the problem of **browser-based knowledge graph construction**: given a query and the starting webpage, the agent will autonomously navigate to a webpage that contains the information that answers the query and output the requested information in a structured format. This problem is relevant in many real-world scenarios, such as improving the productivity of knowledge workers by automating the manual labor of searching, navigating, cleaning the data, and web scraping in situations where websites require complex interactions. Additionally, from a research perspective, this problem can be used to assess the ability of LLMs to act as an information extraction agent that can generalize and navigate the web.

To address such problems, this work introduces **LociGraph**, an AI agent framework for browser-based knowledge graph construction. Figure 1 below shows a demonstration. The source code for both the agent pipeline and the browser extension is publicly released online. The agent pipeline simplifies the webpage into a text format and uses LLM as agents to extract information and decide the next action to take. The browser extension executes the actions on the user’s browser and visualizes the results. Unlike previous works focused on general-purpose web navigation agents, this work focuses on building an agent framework specifically for the task of knowledge graph construction. Additionally, unlike previous works implementing the agents in browser testing tools, this work includes fully implementing a browser extension, further exploring how the latest LLM technology can be developed into a product for a wider audience.

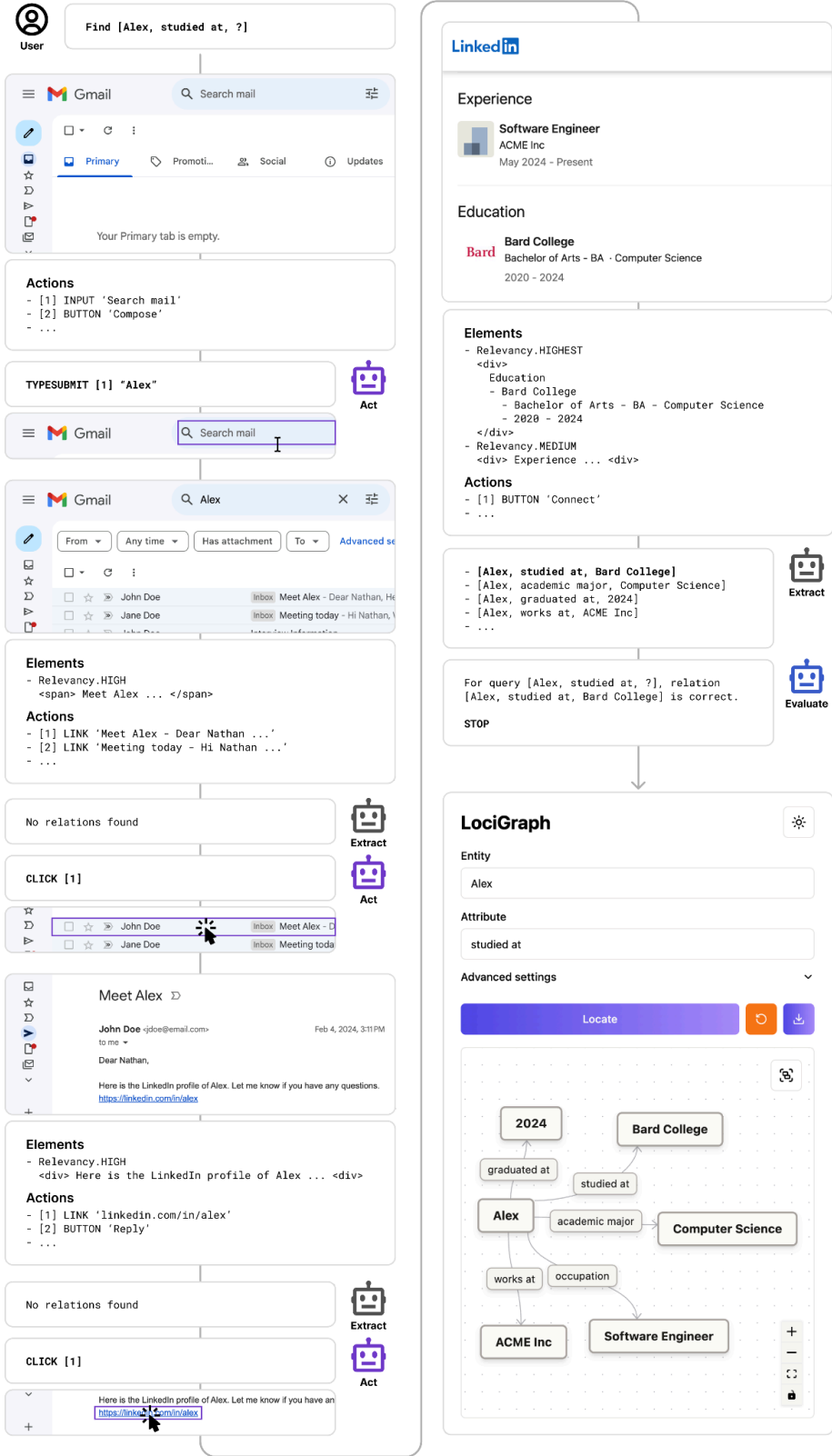


Figure 1: An example of the *browser-based knowledge graph construction* task. The *extract*, *evaluate*, and *act* agents cooperate to answer the query by navigating and extracting information from webpages. A separate browser extension interacts with the browser, providing the webpage content to the agent pipeline.

Within the agent pipeline, the framework implements an *element ranking algorithm* (§4.1.2) to extract information only from the relevant content from the website. A naive way to extract information from a webpage would be simply converting the entire HTML into a text format such as Markdown and searching through the entire text related to the query. As some webpages contain thousands of words, processing the entire text to find a piece of specific information contained in a few words would be a significant waste of resources, which may lead to longer processing times and higher token usage. In order to only process the relevant texts, a rule-based algorithm is designed to rank and filter elements by their relevancy. This algorithm combines whether the content of the element contains synonyms or aliases of the given query and whether the element is located in the main content area of the HTML.

This work leverages the existing research on autonomous web navigation and knowledge graph construction for the implementation. Preliminary evaluation results show that the LLM agents are generally capable of navigating using the browser and constructing a knowledge graph. However, the framework faces challenges when the contents are not directly related to the query. The work suggests ideas to further improve the framework with multimodal input, fine-tuned LLMs, language-model-based webpage understanding, or word similarity-based keyword expansion, discussed in Section 5.

Our contributions are summarized as follows:

- This work introduces the task of *browser-based knowledge graph construction* and an AI agent framework for the task, named LociGraph (§3).
- The framework designs and implements an *element ranking algorithm* that finds the most relevant HTML element for a given query, alongside a custom *textual website representation* for information extraction and navigation (§4.1).
- The framework implements a browser extension that communicates with the framework pipeline and autonomously interacts with the web browser, exploring how end users can benefit from the latest innovation in AI agents (§4.2).

2 Related Work

2.1 Knowledge Graph Construction

Knowledge graph (KG) is a graph-based data model used to organize knowledge, often used in applications like search engines, social networks, recommendation systems, and question-answering systems. [3] Structurally, KG is a network of edges and nodes, each containing a piece of information. Within this network, two nodes connected by an edge are called *knowledge tuples*, which is the minimum unit in KG that represents a single knowledge. Figures 2 below show a typical visualization of KGs and knowledge tuples.

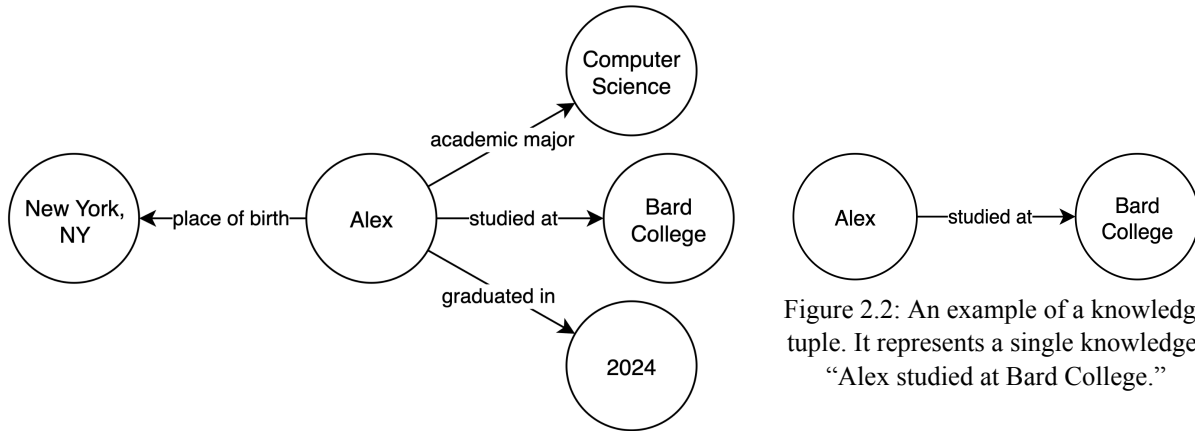


Figure 2.1: An example of KG.

Knowledge graph construction (KGC) is a task where a system is given a data source, such as text documents, databases, and existing knowledge bases, to create an accurate and comprehensive knowledge graph. This task gathered a wide range of research interest over the years due to KG's versatility as a powerful data model, especially in artificial intelligence. KGs can be a great source for training and augmenting generative AI systems as they provide structured and machine-readable data to understand and reason about the world around the system. For instance, answering "Who graduated in 2024?" using two separate data points, "Alex graduated college in 2024" and "Alice went to college from 2019 to 2024," in a linear database, can lead to a complex query. A knowledge graph can answer such a query by simply looking up which nodes are connected to "2024" with the edge "graduated in." As KGs provide the relationship between entities, they can be used in various systems that require an understanding of complex systems of entities. KGs are currently being researched for their potential to be a source of truth for generative AI systems, enabling systems to accurately contextualize the information and draw connections between concepts. [4] This underpins applications like virtual assistants that can accurately answer complex questions by leveraging the KG's interconnected structure.

KGC tasks typically have the following inputs and outputs:

Inputs:

- **Unstructured Data:** Sources of factual information, such as news articles, scientific papers, and webpages. Semi-structured data such as tables, logs, and databases are also used.
- **Existing KG:** Optionally used as a starting point for expansion or refinement of knowledge.
- **Ontology:** Set of rules that define how nodes and edges are created.

Outputs:

- **Knowledge Tuple:** A single knowledge represented as *subject-predicate-object* statements, also known as Resource Description Framework (RDF) triplets. [5] The tuple can be added as an entry in KG databases for querying and browsing tasks, or expressed as vector embeddings for link prediction and reasoning tasks.

Alongside the development of the internet, many systems have been designed to create knowledge graphs at a scale to capture the increasing number of information available on the internet. The following are some of the major developments from KGC research.

Rule-Based Systems The early days of KGC were marked by rule-based systems and manual curation. Pioneering projects like CYC [6] from the 1990s aimed to build a massive, general knowledge base by manually encoding thousands of commonsense facts and rules, such as “You can usually see people’s noses, but not their hearts” or “If you cut a lump of peanut butter in half, each half is also a lump of peanut butter; but if you cut a table in half, neither half is a table.” Another example is YAGO [7], which automatically extracted millions of tuples from Wikipedia using domain-specific rules such as “object, locatedIn, region” or “person, bornInYear, year.” While these projects laid the groundwork, they faced limitations in scalability, adaptability to new domains, and the sheer volume of knowledge that needed to be captured.

Ontology-Driven Systems Following the hand-crafted rules, ontology-driven techniques for KGC emerged. *Ontologies* are formal specifications of concepts and relationships within a particular domain. They define a structured representation of the knowledge within that domain, including the various entities, the properties of those entities, and how they interact with one another. For example, in the domain of higher education, an ontology might include entities such as "student," "professor," "course," and "department," along with relationships such as "enrolled in," "teaches," and "offered by." These ontologies serve as a foundation for KGC tasks as they act as a goal and a guideline for how to construct KGs. Tools like Protégé [8] were developed to aid large-scale KGC tasks by making ontology development easier, but their effectiveness still relied heavily on having manually defined ontologies for each specific domain.

Information Extraction Developments in natural language processing (NLP), especially automation of the *information extraction* (IE) task, accelerated developments of large-scale KGC tasks. IE is a process of extracting structured data from unstructured information, such as news articles. Key sub-tasks within IE include *named entity recognition* (NER), a task of identifying and classifying named entities in text (e.g., “Alex” → person, “New York City” → location, “Bard College” → institution), and *relation extraction* (RE), a task of identifying semantic relationships between entities (e.g., “Alex went to Bard College” → "Alex" [entity] "studied at" [relation] "Bard College" [entity]). Early NER and RE systems like Stanford NER [9] and OpenIE [10] paved the way for automating KGC tasks. Additionally, the development of graph databases like Neo4j and the adoption of the RDF as a standard further fueled advancements in KGC, providing a flexible and interoperable way to store and exchange KG data. [11]

Neural NLP Developments in deep learning and neural networks greatly improved the performance of IE systems. Specifically, followed by breakthroughs in language modeling techniques, pre-trained LLMs have shown great promise in KGC tasks, such as IE or reasoning with complex relations [14], [15]. Notable examples include ERNIE [16], KnowBERT [17], and mREBEL [18]. Trained with massive amounts of real-world text data enabled these models to capture vast amounts of world knowledge embedded within text data at scale, leading to better generalization capabilities for KGC tasks [16].

This work leverages existing work on IE tasks and applies it to a small-scale KGC task. Most related works focus on large-scale KGC tasks, such as web scraping for search engines. In comparison, this work focuses on constructing smaller KGs, as the objective of the framework is extracting data from the non-public web where the amount of data and access are limited compared to the public web (e.g., webpages with varying formats, webpages that block crawler access). To complete the IE tasks, the framework is integrated with a state-of-the-art RE model mREBEL [14], a pre-trained language model based on mBART-50 [19], to extract triplets from texts found on webpages.

Model Input	en_XX Alex graduated from Bard College.
Model Output	tp_XX<triplet> Alex <per> Bard College <loc> educated at</s>
Output Relation	[Alex, educated at, Bard College]

Figure 3: An example of mREBEL relation extraction task. The starting tokens en_XX and tp_XX each correspond to English and Triplets, which are used by the model to denote the language of the text.

In addition to this RE-specific model, the framework employs an LLM agent to extract relations from unconventional text formats, such as HTML tables, which mREBEL might not be able to extract as it is trained on sentences found on Wikipedia. This addition of LLM aims to improve the accuracy and comprehensiveness of the extraction, particularly in identifying complex, nuanced relationships that traditional techniques may not be capable of extracting. By combining the context-understanding capabilities of LLMs with the specialized RE capabilities of mREBEL, the framework aims to construct richer and more reliable knowledge graphs for real-world content on webpages. The implementation details for the relation extraction task are described in Section 4.1.3.

2.2 Autonomous Web Navigation

Autonomous web navigation is a task where an AI agent is given access to the web browser with the goal of completing an objective. The objectives can range from simple tasks (e.g. “Show me airports around me”) to complex tasks with specific requirements and multiple steps (e.g. “Book me a flight to New York departing tomorrow and returning next Wednesday”). The desired output of the task ranges from conversational outputs similar to chatbots [20] to complex website interactions [21].

Autonomous web navigation tasks typically have the following inputs and outputs:

Inputs

- **Objective:** The goal of the agent, usually described using natural language.
- **Observation:** A representation of the webpage, usually a simplified version of the DOM¹ tree in textual format or a screenshot image.
- **Action space:** A series of actions the agent can take at a certain time, usually including clicking, typing, and scrolling, and represented in a similar format as the observation.

¹ Document Object Model (DOM) is an object-oriented representation of a web document [22]. Instead of editing the string content of static HTML code to modify the document, DOM objects provides a programmatic interface to change the structure, style and content of all elements in the document. *DOM tree* is a logical tree representation of the document, constructed with nodes that have children nodes.

Outputs

- **Actions:** Sequence of actions that leads to achieving the objective.
- **Final state:** A state where the objective is completed, usually the target webpage or a natural language response showing the requested objective has been achieved.

With the advent of LLMs, their reasoning and planning capabilities have become a popular backbone of many agent systems in academic and commercial research. Notable examples include MiniWoB++ [23], Mind2Web [24] and WebArena [25] from academia and Adept [26], Multi-On [27] and HyperWrite [28] from startup companies. Beyond LLMs, the recent development of large multimodal models (LMMs) has shown that it is capable of generalizing and understanding complex webpages, opening up new avenues of improving the performances of web navigation agents.

Most of previous work focused on building agents for general-purpose tasks. The most prominent previous work for general-purpose web navigation is Mind2Web [24], a benchmark dataset of 2,350 web navigation tasks from 137 websites that covers 31 task domains. Compared to previous datasets that used simple or simulated websites or mobile applications, Mind2Web used real-world websites with complex layouts and HTML structures, containing more than a thousand HTML elements per page on average. Mind2Web and other web navigation agent research found that representing the observation and the action space into an LLM-compatible format and accurately predicting the next action was a major challenge to building a generalist web agent.

Intent	Task Description
Search	Find the latest news article about Ohio Benefits and share it on Twitter. Find the quickest one-way flight from New York To Paris on June 4 for two adults with one carry-on and one check-in bag in business class.
Save	Create a new portfolio called Tech Stocks and add Microsoft with quality of 1 to it. Find a comedy movie streaming on Netflix and add to watchlist.
Interact	Find and upvote the newest comment about football on the r/news community. Leave a comment on a recipe for chicken parmesan, asking if it can be made in an air fryer.
Other	Get a ten-ride ticket from Washington to New York starting from April 5. Calculate a cost for a package from 10001 to 10023 shipping on April 10.

Figure 4: Example tasks from the Mind2Web dataset, categorized by task intents.

However, these general-purpose web agents are not designed to output structured data. As shown in Figure 4, the output of the task is a different webpage or a response in a conversational manner similar to chatbots, instead of a specific piece of information. This limits the scalability of the information extraction task using such general-purpose agents, as the user has to manually search for the information on the webpage or parse the natural language response.

In contrast to general-purpose agents, WebNav [29], a *focused crawling*² web navigation agent for question answering, has shown that agents can outperform complex inverted-index-based search engines (e.g. Google), suggesting specific-purpose agents are a viable solution for question answering on the web.

² *Focused crawling* is a task of crawling webpages that satisfy specific properties [30]. For example, instead of visiting all link on a webpage, a focused crawler can be configured to only visit pages that are likely to contain information about Alex. This requires the crawler to predict the likelihood of the link leading to a page that satisfy the given properties.

Similarly, instead of building a generalist agent, this work aims to build an agent framework that is specifically designed for *extracting structured data from real-world, non-public webpages*. Implementation details for the web navigation agents are described in Section 4.1.4 and Section 4.1.5.

2.3 Webpage Representation

For both extracting information and prompting LLMs to navigate the web, representing real-world webpages with thousands of HTML elements into a simplified text format has been a challenging task for understanding the web. *Webpage representation* is a task where a system is given the HTML code or the DOM tree of a webpage to represent the webpage in a specific format, such as a simplified text or a list of all interactive elements.

Webpage representation tasks typically have the following inputs and outputs:

Inputs

- **Webpage:** The HTML code or the Document Object Model (DOM) tree of the webpage, optionally including stylesheets (CSS) or the browser state (Window object containing the DOM document).

Outputs

- **Representation:** A text, element, or list of elements represented in a specific format.

Different approaches for textual webpage representation have been developed, including the use of accessibility tree generated by the browser [25], [31], rule-based algorithms [32], graph embeddings [33], [34], [35] and pre-trained language models [36], [37], [38], [39]. In addition to representing the text content of the webpage as simplified text, web navigation agent systems such as FLIN [40] and natbot [41] represented interactive elements as text to use it in LLM prompts.

HTML Code	Textual Representation
<pre><div> Sign in <div> <form> <textarea aria-label="Search"/> <input role="button" value="Search"/> </form> </div> <div>Help</div> </div></pre>	<pre>CURRENT BROWSER CONTENT: ----- <link id=1>Sign in</link> <input id=3 alt="Search"></input> <button id=4>(Google Search)</button> <text id=5>Help</text></pre>

Figure 5: An example of natbot representing HTML code in simplified text. In this representation, HTML tags are generalized (e.g., `<a>` → `<link>`, `<input role="button">` → `<button>`, etc), new `id` attributes are added for identification and accessibility-related attributes (e.g., `alt`, `aria-label`) are used to describe the elements.

As Figure 5 shows, creating a textual representation from raw HTML code requires a lot of conversion and can be highly customized for the task. For instance, natbot generalized all interactive elements into `<link>`, `<button>`, or `<input>` and added the `id` attribute so the LLM could identify which element to use for the next action. Similarly, the agent pipeline implements a customized parser that simplifies the DOM tree and extracts all interactive elements from the page. Additionally, a rule-based *element ranking algorithm* is designed and implemented to rank textual and interactive elements using their HTML tag, attributes, location, and the provided query. The implementation details for the webpage representation and the element ranking algorithm are described in Section 4.1.1 and Section 4.1.2.

3 LociGraph Framework

This section introduces the architecture of **LociGraph**, an agent framework for autonomously navigating and extracting knowledge triplets from non-public webpages. Figure 6 below shows the user interface of the framework, and Figure 7 below shows the architecture of the framework. The overview of the agent pipeline (§3.1) and the browser extension (§3.2) are described in the following sections.

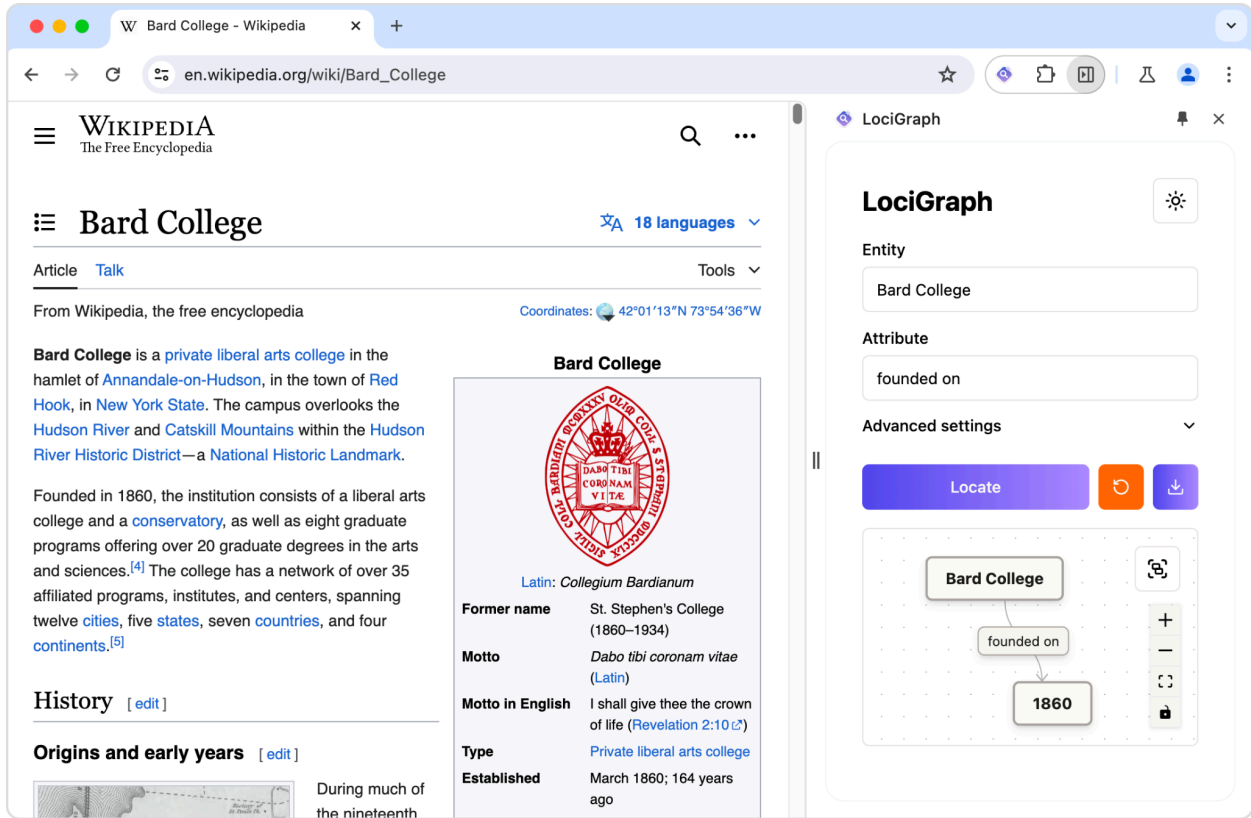


Figure 6: Screenshot of the browser extension of the framework.

The framework is designed for situations where *focused crawling* is required and the webpages can only be accessed from the user’s browser. For example, the framework can automate market research where data exists in platforms that require user login (e.g., email, online community, web application) or exists in an unstructured or irregular format (e.g. company/restaurant websites). The goal of the framework is to autonomously navigate, extract the information requested by the query, and export the results into a CSV file or other machine-readable format.

Our framework has the following inputs and outputs:

Inputs

- **Query:** The relation to complete or verify. Queries can be categorized into 3 objectives:
 - [entity, attribute, value]
Verify whether the *value* is correct for the given entity, attribute, and value.
 - [entity, attribute, ?]
Find the *value* of the given entity and attribute.
 - [entity, ?, ?]
Find all relevant *attributes* and their values for the given entity.

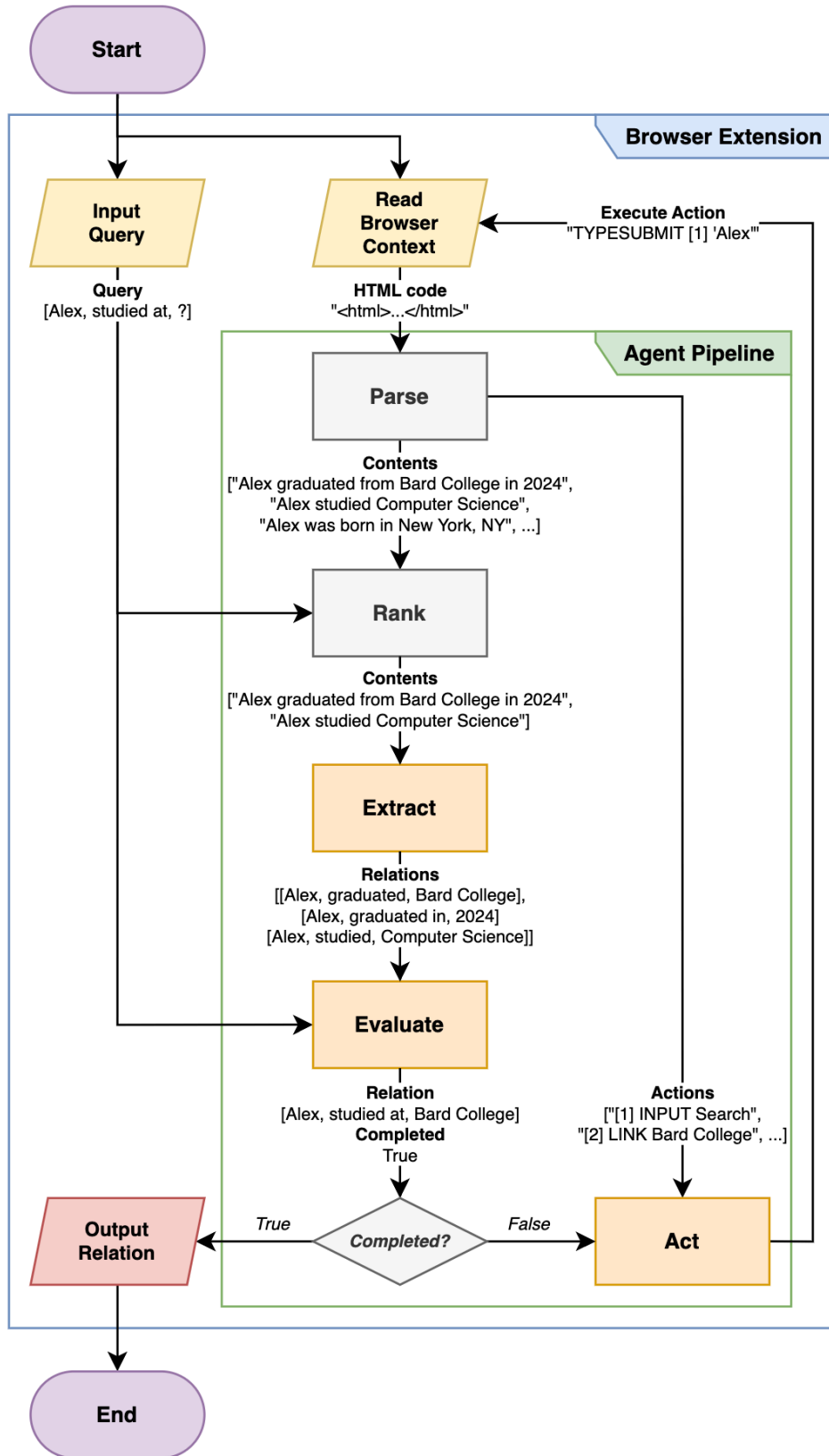


Figure 7: Architecture of the LociGraph framework. Processes colored in orange denote LLM agents. A sequence diagram with more detail is shown in Figure 9 below.

- **Starting webpage:** The webpage to start the navigation.
- **Configuration:** The inference model for LLM agents and whether to enable *continuous mode*³. The user can change these settings on the browser extension.

Outputs

- **Relations:** List of knowledge tuples in RDF triplet format [entity, attribute, value]. The user can browse the relations in the interactive graph visualization or download as CSV file from the browser extension. Empty if the framework couldn't find the relevant information.

The framework leverages state-of-the-art AI models and open-source software development frameworks to implement the framework. The following describes how this work contributes to existing works.

Knowledge Graph Construction As an approach for KGC, this work explores how existing KG research can be applied to small-scale IE tasks on the web. Additionally, this work serves as an exploration of data mining techniques for the non-public web (also known as *deep web*), bypassing the issues of user authentication and crawler blocking websites by directly using the user's browsers.

Autonomous Web Navigation As a specialized approach for autonomous web navigation, this work explores how existing research on LLMs and task-specific models can be combined to create a specialized web navigation agent that performs better with lower costs. Additionally, with the full implementation of the browser extension, this work serves as an exploration of how web navigation agents can be developed as a product instead of a technical proof-of-concept using browser testing tools such as Puppeteer, Selenium, and Playwright.

Website Representation As a task-specific representation, this work explores how representations can be further simplified for LLMs to achieve better performance and lower costs. The element ranking algorithm and the token-optimized website representation format from this framework can be adopted in other web navigation agents. Additionally, this work introduces an extremely fast rule-based DOM simplifying and element ranking algorithm, which can also be used to create general-purpose webpage representation.

3.1 Agent Pipeline

The agent pipeline consists of 5 subtasks: *webpage parsing*, *element ranking*, *relation extraction*, *result evaluation*, and *action prediction*. Each task acts as a data processing or a decision-making process in the pipeline.

Webpage Parsing This task converts the base64 encoded HTML code into a simplified DOM tree ready for element ranking. All non-semantic structures of the tree (e.g., `` tags, elements with no child nodes, etc) are pruned during this process using a rule-based algorithm.

Element Ranking This task ranks static and interactive HTML elements on the current tree and returns the elements that are most relevant to the query. The rank is decided by the text content, attributes, and location of each element, using a rule-based element ranking algorithm.

Relation Extraction This task extracts the knowledge tuples from the text content of the filtered elements, using the mREBEL model and the *extract* LLM agent.

Result Evaluation This task evaluates the completeness and relevance of the knowledge tuple results compared to the query, using the *evaluate* LLM agent. The tuples are ranked by the relevancy to the query, where the tuple answering the query comes first and other related queries follow. If the agent

³ Disabled by default. If enabled, the framework will execute actions without confirmation from the user.

determines the results are satisfactory enough to answer the query, the pipeline will return the current results. Otherwise, the pipeline continues with action prediction.

Action Prediction This task determines the next action (see Figure 8 for possible types of actions) to take to navigate to a webpage containing the relevant content based on current action elements and previous actions, using the *act* LLM agent.

Action	Element Type	Description
CLICK [X]	LINK or BUTTON	Click on the element where <code>id = X</code> .
TYPE [X] 'value'	INPUT	Type <code>value</code> into the element where <code>id = X</code> .
TYPESUBMIT [X] 'value'	INPUT	Type <code>value</code> into the element where <code>id = X</code> and then press Enter.

Figure 8: Overview of the LociGraph action space.

3.2 Browser Extension

The browser extension is the user interface to the agent pipeline and consists of three subtasks: webpage encoding, action execution, and *graph visualization*.

User Interface The browser extension provides control of the agent pipeline and the option to execute the predicted actions. The user can enter the query, configure the inference model, and choose whether to use the continuous mode. The extension is displayed in the side panel of the browser and persists throughout webpage navigation.

Webpage Encoding This task converts the entire DOM tree of the current webpage into a string of base64 encoded HTML code, alongside the URL and the language of the website from the DOM document object.

Action Execution This task executes the next action predicted by the agent pipeline on the current webpage. If continuous mode is enabled, the extension will execute the action autonomously without user intervention.

Graph Visualization This task visualizes the relations extracted so far in an interactive graph on the browser extension. Users can explore how entities are connected with other entities and organize the graph by dragging the nodes or using the auto layout feature.

4 Implementation

This section describes the implementation details for the agent pipeline (§4.1) and the browser extension (§4.2). Figure 9 below shows the sequence diagram of the framework.

4.1 Agent Pipeline

The agent pipeline is implemented as an API server, receiving requests from the browser extension over HTTP connections. This architecture of separating the pipeline from the browser extension minimizes the computational load on end users, enabling the framework to be accessible to all users.

The API server is implemented using `litestar`⁴, a Python Asynchronous Server Gateway Interface (ASGI) framework. The entire agent pipeline is handled by a single API endpoint at `POST /process`, shown in the code excerpt below. The `process_pipeline` function is mapped to the endpoint with the `@post` decorator from `litestar`. See Appendix C.2 for the full OpenAPI specification of the API.

server/app.py

```
@post("/process/", sync_to_thread=True)
def process_pipeline(data: RequestBody, model: str = DEFAULT_MODEL) -> ResponseBody:
    """Process the given extraction query.

    Args:
        data (RequestBody): The extraction query to process.
        model (str, optional): The model to use for extraction. Defaults to DEFAULT_MODEL.

    Returns:
        ResponseBody: The response containing the extracted relations and next action.
    """

    query = data.query # relation query to extract

    # Step 1: Webpage Parsing
    # Parse the base64 webpage data into elements and actions
    scrape = ScrapeEvent(parse(data.data))
    webpage = scrape.data

    # Step 2: Element Ranking
    # Rank and filter relevant elements from webpage
    elements, actions = rank(webpage, query)

    # Step 3: Relation Extraction
    # Extract relations from ranked elements
    extraction = ExtractionEvent(scrape, query, [])

    if len(elements) > 0: # skip if no elements ranked
        extraction.results = extract(elements, query, webpage.title, model)

    # Step 4: Evaluate
    # Evaluate the extraction result
    evaluation = EvaluationEvent([], None, extraction)
    completed: bool = False

    if len(extraction.results) > 0: # skip if no relations are extracted
        completed, relations = evaluate(query, extraction.results, model)
        evaluation.results = relations
```

⁴ <https://litestar.dev/>

```

# Step 5: Action Prediction
# Decide next action based on evaluation result

if not completed: # skip if extraction is complete
    evaluation.next_action = act(
        actions, query, data.previous_actions, webpage.url, webpage.title, model
    )

# Step 6: Respond
# Return the extracted relations and next action to browser
return evaluation.getresponse()

```

4.1.1 Webpage Parsing

This task extensively uses `lxml`⁵, a Python XML/HTML processing library that enables parsing HTML code into DOM objects and make modifications. The `parse` module handles the parsing, cleaning, and simplifying the webpage, as shown in the code excerpt below. The following sections describe how each subtasks are implemented.

server/parse.py

```

def parse(data: WebpageData) -> ParsedWebpageData:
    """Parse the elements and actions from the given webpage data and simplify the DOM tree.

    Args:
        data (WebpageData): Webpage data to parse

    Returns:
        ParsedWebpageData: Parsed webpage data
    """

    html_text = b64decode(data.htmlBase64).decode("utf-8")

    # parse the HTML using lxml
    html: HtmlElement = fromstring(html_text, base_url=data.url)
    tree: _ElementTree = ElementTree(html)

    # parse the title of the webpage
    title = " ".join(html.xpath("//title/text()"))
    title = title if len(title) > 0 else None

    # parse CSS rules to AST nodes
    global all_rules
    all_rules = parse_css_to_ast(html.xpath("//style"))

    # clean and simplify the DOM tree
    html, tree = flag_noise_elements(html, tree)
    html, tree = flag_action_elements(html, tree)
    html, tree = remove_noise_elements(html, tree)
    html, tree = replace_aria_roles(html, tree)
    actions = get_action_elements(html, tree)
    html, tree = simplify_html(html, tree)

    return ParsedWebpageData(
        data.url, data.htmlBase64, data.imageBase64, data.language, title, html, tree, actions
    )

```

⁵ <https://lxml.de/>

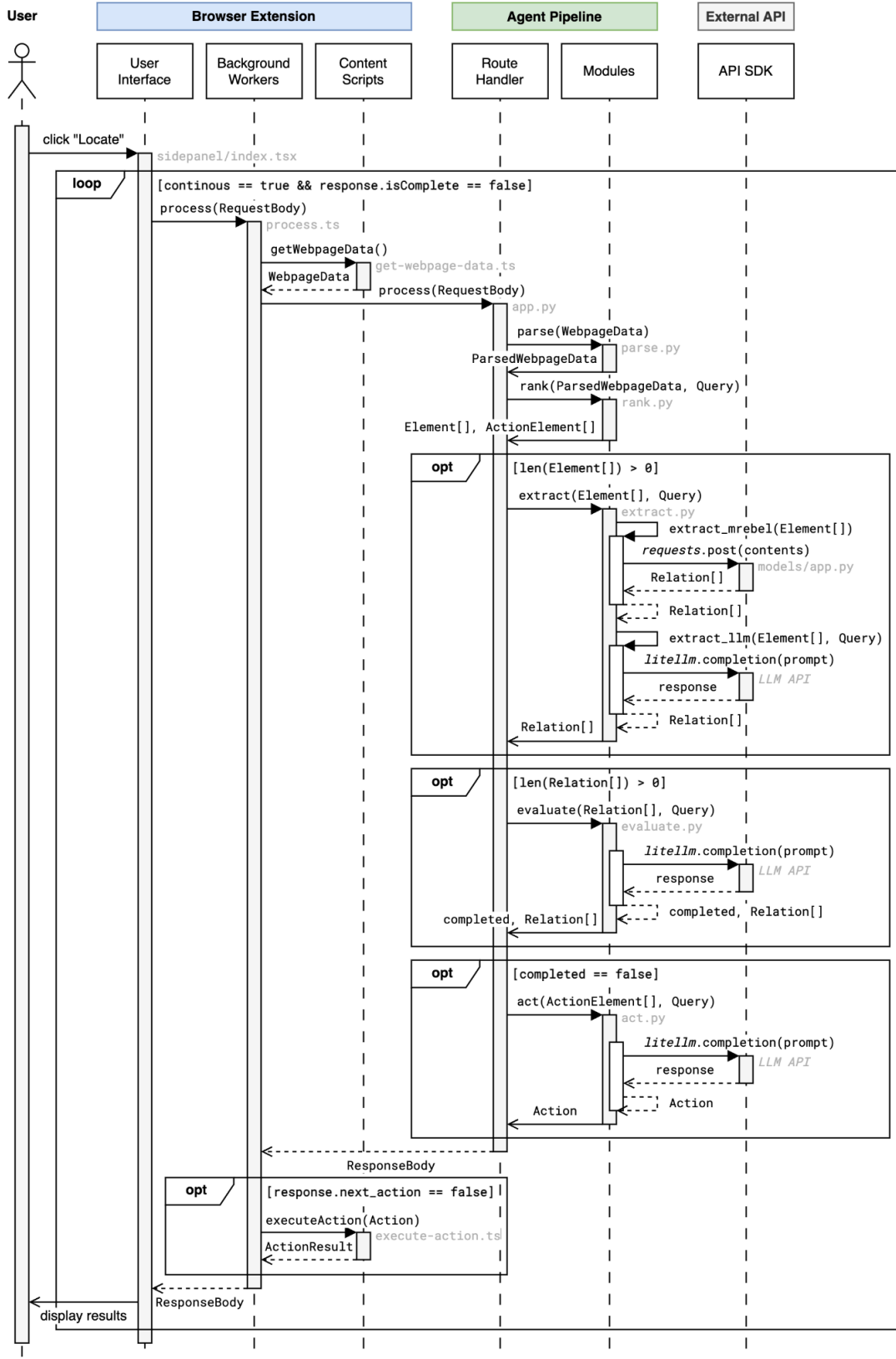


Figure 9: Sequence diagram of the LociGraph framework. The browser extension, agent pipeline and external API communicate with each other via REST API over HTTP requests.

flag_noise_elements

This function flags all elements that will never contain text contents relevant for the relation extraction as noise, with a combination of rules that filter elements with HTML attributes, CSS classes, and tags using XPath⁶ queries on the DOM object.

Attribute Filtering Elements that contain attributes that typically indicate hidden or non-visible content are flagged as noise. For instance, class attributes containing values such as "hidden" and "invisible" are commonly used to hide elements in popular CSS frameworks such as TailwindCSS⁷ and Bootstrap⁸. Similarly, style attributes with values like "display: none" or "visibility: hidden" are flagged as the website does not intend the element to be displayed. The elements are selected using the XPath function contains(), as shown in the example below.

```
XPath //*[contains(@class, 'hidden')]
HTML <body>
    <div class="hidden p-4"> Invisible </div>
</body>
Result <div class="hidden p-4"> Invisible </div>
```

CSS Selector Filtering While attribute filtering can detect whether an element is displayed or not, it is not capable of detecting if the style of the element is defined outside of the element. This filtering method compensates for the limitation of attribute filtering by flagging elements that contain CSS selectors where the style indicates the element is invisible. First, the function finds all selectors that have styles that make the element invisible from the CSS code extracted from <style> tags on the page. In addition to the CSS style used in the attribute filtering, additional styles such as opacity: 0, transform: scale(0), and height: 0 are used to filter the selectors. Then, all elements with such selectors are flagged using lxml.cssselect, as shown in the example below.

```
HTML <html>
    <head>
        <style> .hide { display: none } #transparent { opacity: 0 } </style>
    </head>
    <body>
        <div class="hide"> Invisible </div>
        <div id="transparent"> Invisible </div>
    </body>
</html>
Selectors [".hide", "#transparent"]
Result <div class="hide"> Invisible </div>
    <div id="transparent"> Invisible </div>
```

Additionally, filtering the selectors and filtering the elements with the selectors are computed in multiple threads using the concurrent.futures module from Python, as modern websites often use compiled CSS, which contains tens of thousands of selectors that map to thousands of elements.

⁶ XPath, also known as *XML Path Language*, is a powerful syntax to address, represent and test parts of an XML or HTML document. [42] For example, evaluating XPath //div returns all <div> elements in the document, /div/a[0] returns the first first <a> tag that is a direct child of the <div> element on root, //body/string() returns the entire text content of the <body> element including all child elements.

⁷ <https://tailwindcss.com/>

⁸ <https://getbootstrap.com/>

Tag Filtering Certain HTML tags, by their nature, do not contain textual content or represent structural elements that are not relevant for relation extraction. This method flags elements with tags like <head>, <script>, <style>, <iframe>, <svg> and others, to exclude them from further processing. Additionally, all HTML comment elements are dropped from the tree to reduce the number of elements to process.

Flagging Noise Elements Instead of dropping elements from the tree as soon as they are filtered, elements are flagged first and dropped later in the process. This is done to preserve the original DOM tree hierarchy for action elements. As the framework requires the original XPath to the action element in order to execute the action in the browser extension, modifying the DOM tree before identifying the action elements can lead to two different XPaths for the same action element. Within the framework, a custom attribute `locigraph-type` is used as a flag to indicate whether an element has been processed or not. Elements and their children elements identified in this function will have the value "NOISE" for the custom attribute, indicating that these elements will be removed in subsequent processing, shown in the example below.

```
HTML <body>
Before   <div class="hidden p-4">
          Invisible
          <span> Child </span>
        </div>
        </body>

HTML <body>
After    <div class="hidden p-4" locigraph-type="NOISE">
          Invisible
          <span locigraph-type="NOISE"> Child </span>
        </div>
        </body>
```

flag_action_elements

Similar to `flag_noise_elements`, this function flags all interactive elements as *action elements* with their respective types. As shown in Figure 8, the framework uses LINK, BUTTON and INPUT categorize the type of action items.

LINK Extraction Tag, CSS selector and attribute filtering is used to extract LINK elements. For *tag filtering*, XPath query `//a[string-length(text()) > 0]` is used to find all <a> elements with text content inside. For *CSS selector filtering*, the style cursor: pointer is used to find all elements that visually indicate clickability. Most web navigation agents do not filter these elements that are actually interactive but don't appear to be interactive on the HTML code. For *attribute filtering*, the `aria-role` attribute with value "link" is used to find elements that indicate hyperlink for screen readers.

BUTTON Extraction Tag and attribute filtering is used to extract BUTTON elements. For *tag filtering*, XPath query `//button[string-length(text()) > 0]` is used to find the usual <button> elements. Additionally, a combination of tag and attribute filtering is used for <input> elements that display as buttons with XPath `//input[@type=' {input_type}']` where `input_type` is button, reset and submit. For *attribute filtering*, all elements that have any of `onclick`, `ondblclick`, `onmousedown` and `onmouseup`, and `aria-role` attribute with value "button" is used to find additional button elements.

INPUT Extraction Similar to BUTTON, tag and attribute filtering is used to extract LINK elements. For *tag filtering*, XPath query `//textarea` is used to find <textarea> elements. Additionally, XPath `//input[not(@type='hidden' or @type='{input_type}']` is used to find all other <input> elements that are

not hidden nor buttons. For *attribute filtering*, the `aria-role` attribute with value “textbox” or “searchbox” is used to find elements that indicate inputs that can take text inputs.

Flagging Action Elements After all action elements are found, types of the action elements are added to the custom attribute similar to the previous function. Additionally, the original XPath to the element is added in a custom attribute `locigraph-xpath`, which will be used later in action execution in the browser extension, shown in an example below.

```
HTML <body>
Before  <div>
        <a href="https://example.com"> Link </a>
      </div>
      <div onclick="clickHandler()"> Button </div>
      <form>
        <input type="search" />
      </form>
    </body>
```

```
HTML <body>
After   <div>
        <a href="https://example.com"
          locigraph-type="LINK"
          locigraph-xpath="//body/div[0]/a"
        > Link </a>
      </div>
      <div onclick="clickHandler()"
        locigraph-type="BUTTON"
        locigraph-xpath="//body/div[1]"
      > Button </div>
      <form>
        <input type="search"
          locigraph-type="INPUT"
          locigraph-xpath="//body/form/input"
        />
      </form>
    </body>
```

remove_noise_elements

Following the flagging of noise elements in `flag_noise_elements` and flagging and preserving the XPaths of action elements in `flag_action_elements`, this function drops all elements flagged as noise from the DOM tree.

Noise Removal Similar to attribute filtering in previous functions, XPath `//*[@locigraph-type='NOISE']` is used to select all elements flagged as noise. For each selected element, the `drop_tree()` method from `lxml.html.HtmlElement` is used to remove the element and its entire subtree from the DOM tree, shown in the example below.

```

HTML <body>
Before   <div class="hidden p-4" locigraph-type="NOISE">
          Invisible
          <span locigraph-type="NOISE"> Child </span>
        </div>
        <span> Visible </span>
      </body>

HTML <body>
After    <span> Visible </span>
      </body>

```

replace_aria_roles

Denoted in the `aria-role` attribute with values such as “banner”, “navigation” and “region”, landmark roles communicate that the element is an important subsection of a page to browser accessibility engines [43]. While some webpages use semantic tags such as `<header>` to create the header section of the webpage, some use generic tags with landmark roles such as `<div aria-role="banner">`. As the framework uses the location of the element to rank its relevancy (e.g. content within `<section>` is likely to be more relevant than content within `<aside>`; see Section 4.1.2), this function identifies and replaces the tags in elements with ARIA landmark roles into an equivalent HTML tag using *attribute filtering*, shown in the example below.

```

HTML <body>
Before   <div aria-role="banner"> Header content </div>
          <div aria-role="complementary"> Aside content </span>
          <div aria-role="region"> Section content </span>
        </body>

HTML <body>
After    <header> Header content </header>
          <aside> Aside content </aside>
          <section> Section content </section>
        </body>

```

get_action_elements

This function serves as the final step in the webpage parsing stage. It focuses on processing the previously flagged action elements and creating `ActionElement` objects, which encapsulate essential information for subsequent action prediction tasks.

ActionElement Object Creation The function begins by finding elements based on their assigned `locigraph-type` attribute values "INPUT", "BUTTON", and "LINK". For each element, an `ActionElement` object is created with metadata about the element, including:

- `xpath`: The element's full XPath within the original HTML document, retrieved from the `locigraph-xpath` attribute.
- `modified_xpath`: The element's full XPath within the latest DOM tree, retrieved from current DOM object. This includes changes from noise element removal and ARIA landmark role tag replacement.
- `html_element`: The `lxml.html.HtmlElement` object, containing the DOM element details such as tag, attributes, children and parent nodes.
- `content`: The textual content for LINK, BUTTON elements and the input value for INPUT elements.

- **details:** A dictionary containing additional attributes of the element (e.g., { href: "https://..." } for LINK elements, { placeholder: "Search Wikipedia", type: "search" } for INPUT elements).
- **type:** The element type ("INPUT", "BUTTON", or "LINK").

The function returns a list of ActionElement objects, representing all the interactive elements identified on the webpage, shown in the example below. These objects are used in subsequent element ranking and action prediction processes.

```
HTML <body>
  <header>
    <a href="https://example.com"
      locigraph-type="LINK"
      locigraph-xpath="//body/div[0]/a"
    > Link </a>
  </header>
  <aside>
    <input type="search"
      locigraph-type="INPUT"
      locigraph-xpath="//body/form/input"
      value="existing value"
      placeholder="Search"
    />
  </form>
</body>

Result [
  ActionElement(
    xpath="//body/div[0]/a"
    modified_xpath="//body/header/a"
    html_element=<HtmlElement a>
    content="Link"
    details={ href: "https://example.com" }
    type="LINK"
  ),
  ActionElement(
    xpath="//body/form/input"
    modified_xpath="//body/header/a"
    html_element=<HtmlElement input>
    content="existing value"
    details={ placeholder: "Search" }
    type="INPUT"
  )
]
```

simplify_html

This function optimizes the DOM tree by eliminating unnecessary tags and attributes. This simplification process reduces the complexity of the document and flattens the tree, unifying text elements into fewer elements and reducing the size of the tree, making it more optimal for the relation extraction task.

aria-hidden Content Consolidation Elements hidden from the accessibility tree are merged with the parent, as they are likely to contain visual content. Elements are found using the XPath

//*[aria-hidden='true'] and the child nodes are merged with the parent node by using the `drop_tag()` method from the `lxml.html.HtmlElement` class, shown in the example below.

```
HTML <body>
Before   <div aria-hidden='true'>
          Text Content
          <span> Additional content </span>
        </div>
</body>

HTML <body>
After    Text Content
          <span> Additional content </span>
</body>
```

Attribute Removal To further reduce the size and the complexity of the DOM tree, all attributes from all remaining elements are removed. As all of the following steps are about textual representation, element attributes are no longer relevant.

Empty Element Removal Elements that have no textual content are replaced with a single space character. These elements contribute no semantic value and can be safely eliminated without affecting the meaning or structure of the document. The elements are replaced with a space character to prevent text nodes being joined together without a whitespace in between, shown in the example below. This step is performed iteratively until no more empty elements are found.

Cosmetic Tag Removal Tags such as ``, `<i>` and `<u>` are primarily used for styling or presentation purposes and have no inherent semantic meaning. As these tags do not contribute to the content of the element, the child nodes of elements with these tags are merged with the parent node, shown in the example below.

```
HTML <div>
Before   Hello
          <span></span>
          <b>World</b>
        </div>

HTML <div>
After    Hello World
        </div>
```

Table Representation `<table>` elements often have a lot of child elements with table rows and cells. If the table is converted into textual format with a method like *tree flattening* shown below, the semantic information of the columns and cells will be lost. To prevent this, contents of `<table>` elements are converted into Markdown-style table format, as shown in the example below.

```
HTML <table>
Before   <caption>Title</caption>
          <thead>
            <tr><th>H</th> <td>1</td> <td>2</td></tr>
          </thead>
          <tbody>
            <tr><th>A</th> <td>1</td> <td>2</td></tr>
            <tr><th>B</th> <td>1</td> <td>2</td></tr>
```

```

        </tbody>
    </table>
HTML <table>
After  [table]
       [Title]
       H | 1 | 2
       -----
       A | 1 | 2
       B | 1 | 2
    </table>

```

Tree Flattening Elements that have only one child element node without a child text node creates deeply nested tree, creating complexity without adding semantic value. In such cases, the parent element's tag is deemed redundant and can be removed, effectively merging the child element into the parent's position in the DOM tree. This process is also repeated iteratively until no further consolidation is possible.

```

HTML <div><div><div><div><div>
Before Hello<div>World</div>
      </div></div></div></div></div>

HTML <div>
After  Hello <div>World</div>
      </div>

```

List Representation Similar to `<table>` elements, list elements such as `` or `` can benefit from text content replacements. During the ranking process, if one of the list item contains the keyword, the ranking algorithm will only return the matching list item, which often results in loss of context as other elements in the list are not included. To prevent this, contents of list elements are converted into Markdown-style list format, as shown in the example below.

```

HTML <div>
Before <ol>
      <li>foo</li>
      <li>bar</li>
    </ol>
    <ul>
      <li>foo</li>
      <li>bar</li>
    </ul>
  </div>

HTML <div>
After  <ol>
      1. foo
      2. bar
    </ol>
    <ul>
      - foo
      - bar
    </ul>
  </div>

```

4.1.2 Element Ranking

Using the simplified DOM tree and the extracted `ActionElement` objects, the `rank` module handles ranking and filtering relevant text and action elements that will be used by *extract* and *act* agents later in the pipeline, shown in the code excerpt below. The following sections describe how each subtasks are implemented.

server/rank.py

```
def rank(
    data: ParsedWebpageData, query: RelationQuery
) -> tuple[list[Element], list[ActionElement]]:
    """Rank and return relevant elements based on the given keywords.

    Args:
        data (ParsedWebpageData): The parsed webpage data.
        query (RelationQuery): The relation query to rank elements.

    Returns:
        list[Element], list[ActionElement]: The ranked elements and action elements.
    """

    keywords: list[tuple[str, Relevancy]] = get_top_keywords(query, data.title)

    elements = rank_elements(data, keywords)
    action_elements = rank_action_elements(data, keywords)

    return elements, action_elements
```

get_top_keywords

This function returns the top K keywords and its relevancy score⁹ used to rank and filter elements, from the given query and webpage title. For example, if the query is [Alex, studied at, ?], elements that contain contents about Alex or his education must be ranked higher than other elements. Although we can look for contents about Alex by directly searching for the name of the entity “Alex”, contents about where Alex went to school is much harder to find, as searching with the exact term “studied at” is unlikely to match the content on the webpage. In order to increase the chance of finding a match, these keywords must be expanded to its synonyms and aliases. The following sections describe how these keywords are expanded and ranked.

Using Entity Name By default, the name of the entity is added to the keyword list with the HIGHEST relevancy. But if the webpage title contains the name of the entity, it is highly likely that the entire webpage content is relevant to the entity and many matches will be found. In such cases, in order to emphasize the keywords related to the attribute, the relevancy will be set to HIGH.

Expansion with Wikidata Properties Wikidata¹⁰ is an online knowledge base containing more than 100 million items (i.e. entities) and 11,000 properties (i.e. attributes) connecting each other, creating more than 1.5 billion statements (i.e. groups of relations). [44], [45] The framework leverages this collection of properties and its aliases for expanding the attribute keyword. For example, for attribute “studied at”, Wikidata property contains 21 other aliases such as “alma mater”, “educated at”, “college attended” and “graduated from.” As these aliases are very effective synonyms, all of the aliases are added to the keyword list with the relevancy of HIGHEST.

⁹ Relevancy score can be one of the following: LOW (0.25), MEDIUM (0.5), HIGH (0.75), HIGHEST (1.0)

¹⁰ <https://www.wikidata.org/>

Expansion with Wordnet Wordnet is a lexical reference system organized by concepts. In other words, Wordnet is similar to a thesaurus that groups words together based on their concepts, which is useful to find synonyms and related words. [46] The framework leverages this data to expand the attribute keyword beyond Wikidata aliases. From Wordnet, first degree synonyms of the attribute are added with the relevancy of HIGH, and second degree synonyms are added with LOW. Second degree synonyms are rarely used as their relevancy is lower than other keywords, but it can be useful in situations where there are no aliases and there are only a few first degree synonyms.

Top-K Selection Instead of using all keywords generated to filter elements, the function only returns the top K keywords. If the keyword list has n keywords with HIGHEST relevancy and $n > 25$, K will be equal to n . Otherwise, K defaults to 25. The keywords are sorted by relevancy in a descending order and the top K keywords are returned.

filter_elements

This function ranks elements that are most likely to contain relevant information for a given query. This function leverages XPath queries with regular expression and a rule-based element ranking algorithm.

XPath Query Generation Based on the expanded keywords, the function constructs XPath queries to target elements containing these keywords within their text content. For example, if the keywords are expanded to ["Alex", "studied at"], the following regex-based XPath query will be generated: `//*[re.test(text(), 'Alex', 'i')] | /*[re.test(text(), 'studied +at', 'i')]`. This query filters elements where its text content matches they keyword, insensitive to case and whitespace. These XPaths are grouped by keywords with the same relevancy score, returning single XPath query per relevancy score.

Element Object Creation The function evaluates each XPath query on the DOM tree, retrieving a list of matching elements. For each element, an Element object is created, including:

- `xpath`: The element's full XPath within the latest DOM tree, retrieved from current DOM object. This includes changes from simplifying the DOM tree.
- `html_element`: The `lxml.html.HtmlElement` object.
- `content`: The textual representation of all child nodes of the element.
- `relevance`: A dictionary containing content and location relevancy score.
 - *Content Relevancy Score*: This score reflects the importance of the keywords found within the element's text content, as determined by the relevancy level assigned during XPath query generation.
 - *Location Relevancy Score*: This score considers the element's position within the DOM tree. Elements are ranked HIGHEST if its ancestor include `<article>` or `<section>` elements, HIGH for `<main>`, MEDIUM for `<header>`, `<footer>` and all other elements, and LOW for `<aside>` and `<nav>`.

The function returns a sorted list of Element objects, with most relevant element appearing first, shown in the example below. This ranking is based on the product of content and location relevancy scores, ensuring that elements likely to contain the desired information are prioritized for relation extraction.

filter_action_elements

Similar to `filter_elements` function, this function ranks previously identified ActionElement objects using a similar rule-based algorithm to prioritize actions that are more likely to lead to the desired information.

Search Input Prioritization The function first checks if any of the action elements are search inputs, by checking whether any of the attributes include the term "search". If found, these elements are assigned the HIGHEST relevancy score, as they can potentially directly lead the agent to a page with relevant content.

Relevance Scoring For all other elements, their *content* and *location relevancy score* is computed in the same way as above. As all ActionElement objects are already created by the parse module, the elements are simply checked whether its content contains the keyword. If the element does not contain any of the provided keywords, its content relevancy score defaults to LOW. This ensures that all action elements are considered but prioritized based on their potential relevance to the query.

This function returns a sorted list of ActionElement objects, with the most relevant actions appearing first. Additionally, each action element is assigned a unique id attribute that will be used by the *act* agent.

4.1.3 Relation Extraction

Using the text content found from the ranked elements, the extract module converts natural language text into knowledge triplets using the RE model mREBEL¹¹ and the *extract* LLM agent. As both mREBEL and LLMs are served via APIs, requests for both models are processed in parallel using the `concurrent.futures` module from Python, as shown in the code excerpt below.

server/extract.py

```
def extract(
    elements: list[Element],
    query: RelationQuery,
    title: str | None,
    model_id: str = DEFAULT_MODEL,
    mock_response: str | None = read_mock_response("data/mock_response_extract.txt"),
) -> list[Relation]:
    """Extract relation triplets from the given elements.

    Args:
        elements (list[Element]): List of elements to extract relations from.
        query (RelationQuery): The query of relations to extract.
        title (str): The title of the webpage.
        model_id (str): The ID of the LLM model to use for extraction.
        mock_response (str): The mock response to use for testing.

    Returns:
        list[Relation]: List of extracted relations triplets or empty list if the
        extraction failed.
    """

    results: list[Relation] = []

    # extract relations with mREBEL and LLMs concurrently
    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = [
            executor.submit(extract_mrebel, elements, title),
            executor.submit(extract_llm, elements, query, title, model_id, mock_response),
        ]

        for future in concurrent.futures.as_completed(futures):
            results.extend(future.result())
```

¹¹ <https://huggingface.co/Babelscape/mrebel-large>

return results

extract_mrebel

This function handles RE using the mREBEL model. The mREBEL model is served from a separate `litestar` instance, in order to keep the resource usage of the main pipeline optimal. This API-based architecture enables other RE model to be easily added or served from the cloud.

Text Preparation The function begins by extracting the text content from the provided `Element` objects. Text content from elements, filtered with content (i.e. not `None`) and a combined relevancy score greater than 0.5, are concatenated into a single string, prepared for the mREBEL API.

mREBEL API Interaction The concatenated text is sent as a `POST` request to the `/extract/` endpoint of the mREBEL API (See Appendix C.2 for the implementation). This endpoint is responsible for processing the input text and applying the mREBEL model to identify and extract relation triplets into `Relation` objects.

extract_llm

This function offers an alternative approach to relation extraction, using LLMs to identify and extract relation triplets from webpage elements. A prompt template (see Appendix A.2 for the full prompt including the in-context learning examples) is used to guide the LLM to generate the desired output, shown in the example below. The LLM agents, *extract*, *evaluate* and *act* agents, are implemented by generating a the prompt using the templates and sending the prompts to the LLM APIs using `litellm`¹², a unified SDK for LLM APIs.

Prompt Generation The prompt is generated by the prompt module, which contains the instruction and in-context learning examples with chain-of-thought prompting. *In-context learning* (ICL), also known as *few-shot prompting*, is an LLM prompting technique where the model is provided with instructions and a few examples of inputs and outputs [47]. *Chain-of-thought* (CoT) is another prompting technique where the model is shown examples of intermediate reasoning steps, eliciting the LLM to generate a response based on reasoning [48]. These techniques are used to extract elements with precision and in an easily parsable format. The prompt ends with the first sentence of the CoT prompt, making the LLM start generating from reasoning, reducing the risk of hallucination. The excerpt of the *extract* prompt template is shown below, ICL examples highlighted in green and CoT prompts highlighted in bold.

```
Elements [
  Element(
    ...
    content="Education
    - Bard College
      - BA in Computer Science
      - 2020-2024"
    relevance={ location: Relevancy.HIGH, content: Relevancy.HIGHEST }
  ),
  Element(
    ...
    content="Alex works at ACME Inc as a Software Engineer."
    relevance={ location: Relevancy.HIGHEST, content: Relevancy.MEDIUM }
```

¹² <https://www.litellm.ai/>

)
]

Prompt You are tasked to extract all relation triplets from a webpage. Imagine you are imitating humans using a web browser to find information, step by step.

The following are some examples:

Title: `About Alex | alex.com`

Content:

...

Born

January 1, 2000

New York City, NY, US

...

Query: [Alex, date of birth, ?]

Reasoning: Let's think step by step. We need to find the date of birth of Alex. The content provides the birthday January 1st, 2000. Since the content provides the date of birth, we should extract the relation [Alex, date of birth, January 1st, 2000]. Additionally, the content provides the birthplace New York City. Since this is related to Alex, we should extract the relation [Alex, place of birth, New York City] as an additional relation.

Output:

Query relations:

- [Alex, date of birth, January 1, 2000]

Additional relations:

- [Alex, place of birth, New York City]

=====

((2 examples omitted))

Page title: Alex | LinkedIn

Content:

...

Education

- Bard College
- BA in Computer Science
- 2020-2024

Alex works at ACME Inc as a Software Engineer.

...

Query: [Alex, studied at, ?]

Reasoning: Let's think step by step.

Response Parsing As the LLM responses are in plain text, the relations are parsed from the text using regular expression matches. All strings that match - [entity, attribute, value] format is converted into a Relation object by the parser in the prompt module.

4.1.4 Result Evaluation

Instead of directly using the output from the *extract* agent, the framework evaluates the results to confirm whether the extracted relations answer the given query and decide whether further action is required. The implementation of LLM APIs using the `completion()` from `litellm` is shown in the code excerpt below.

server/evaluate.py

```
from litellm import completion

def evaluate(
    query: RelationQuery,
    results: list[Relation],
    model_id: str = DEFAULT_MODEL,
    mock_response: str | None = read_mock_response("data/mock_response_evaluate.txt"),
) -> tuple[bool, list[Relation]]:
    """Evaluate the extracted relations and determine if the query is completed.

    Args:
        query (RelationQuery): The query of relations to evaluate.
        results (list[Relation]): The list of relations to evaluate.
        model_id (str): The ID of the LLM model to use for evaluation.
        mock_response (str): The mock response to use for testing.

    Returns:
        bool, list[Relation]: A boolean indicating whether the query is completed and
        a list of evaluated relations.
    """

    try:
        response = completion(
            messages=generate_evaluate_prompt(query, results),
            model=model_id,
            mock_response=mock_response,
            stop="=====",
            logger_fn=litellm_logger,
        )

        response_content = response["choices"][0]["message"]["content"]

        is_complete, relations = parse_evaluate_response(response_content)

        return is_complete, relations

    except Exception as e:
        log.error(f"Failed to evaluate {len(results)} relations with LLM `{model_id}`")
        log.exception(e)

        raise exceptions.HTTPException(
            status_code=500,
            detail=f"Couldn't evaluate current progress. {error.CHECK_LLM}",
        )
```

evaluate

Prompt Generation Using prompt techniques similar to the *extract* agent, the *evaluate* agent is implemented using LLM. The *evaluate* prompt template used by the prompt module is shown below.

Relations [

```
Relation(entity="Alex", attribute="studied at" value="Bard College"),
Relation(entity="Alex", attribute="studied" value="Computer Science"),
Relation(entity="Alex", attribute="graduated in" value="2024"),
Relation(entity="Alex", attribute="works at" value="ACME Inc"),
Relation(entity="Alex", attribute="job title" value="Software Engineer")
```

]

Prompt You are tasked to evaluate relation extraction results for the given query. Imagine you are imitating humans using a web browser to achieve an objective, step by step.

The following are some examples:

Query: [Alex, date of birth, ?]

Extraction results:

- [Alex, birthday, January 1, 2000]
- [Alex, born on, 2000]
- [Alex, born in, New York]

Reasoning: Let's think step by step. We need to find the date of birth of Alex. [Alex, birthday, January 1st, 2000] provides the date of birth as `January 1st, 2000`, therefore this is correct. [Alex, born on, 2000] only provides the year compared to the previous relation, therefore this is incorrect. [Alex, born in, New York] is not related to date of birth, but it is related to the query, therefore this is relevant. Since at least one extraction result is correct, we should `STOP`.

Answer: STOP

Correct relation:

- [Alex, date of birth, January 1, 2000]

Relevant relation:

- [Alex, place of birth, New York]

=====

((2 examples omitted))

Query: [Alex, studied at, ?]

Extraction results:

- [Alex, studied at, Bard College]
- [Alex, studied, Computer Science]
- [Alex, graduated in, 2024]
- [Alex, works at, ACME Inc]
- [Alex, job title, Software Engineer]

Reasoning: Let's think step by step.

Response Parsing The response of this prompt has three components: *answer*, *correct relation* and *relevant relations*. The *answer* can be either "STOP", indicating that the extracted relations are sufficient to complete the query, or "CONTINUE", suggesting that further navigation actions are required to find more relevant contents. Additionally, the LLM provides the *correct relation* and the list of *relevant relations*, by classifying the relevant relation and renaming the attributes if needed (e.g. if the query attribute was "employer" and the extracted relation is [Alex, works at, ACME Inc], it will rename the attribute to the

query attribute and return [Alex, employer, ACME Inc]). For *answer*, the parser checks whether the string "answer: stop" or "answer: continue" is contained in the lowercase-converted response string and returns True for STOP, and False for CONTINUE. For relations, same as the *extract* response parser, all strings that match - [entity, attribute, value] format is converted into a Relation object.

If the *evaluate* agent returns the boolean True, indicating the relation that completes the query has been extracted, the pipeline is complete and returns all extracted relations as the API response. Otherwise, the pipeline continues to the *act* agent to decide the next action.

4.1.5 Action Prediction

This task is the last processing step of the pipeline, handling situations where the agents couldn't extract the desired relation from the current webpage. The module implementation is similar to the *evaluate* module, shown in Appendix C.2.

act

Using the previously extracted ActionElement objects, query, previous actions, and the URL and the title of the webpage, the *act* agent predicts a single action that is most likely to lead to more relevant content.

Prompt Generation The *act* prompt is similar to previous prompts, shown in the example below. Instead of prompting the agent to find the query in the [entity, attribute, value] format, the query is generalized into a natural language *objective* to elicit more creative reasoning from the LLM agent, as shown in Figure 10 below. The instruction and the structure of the prompt for *act*, *extract* and *evaluate* was inspired by previous works on web navigation agents, including natbot [41] and Mind2Web [24].

Query	Objective
[entity, attribute, value]	Verify whether entity `entity` has attribute `attribute` with the value `value`.
[entity, attribute, ?]	Find the value of attribute `attribute` of entity `entity`.
[entity, ?, ?]	Find all attribute of entity `entity`.

Figure 10: Overview of the query objective generation.

```

Actions [
  ActionElement(
    ...
    content=""
    details={ placeholder: "Search Google" }
    relevance={ content: Relevancy.HIGHEST }
    type="INPUT"
    id=1
  ),
  ActionElement(
    ...
    content="I'm feeling lucky"
    relevance={ content: Relevancy.LOW, location: Relevancy.HIGH }
    type="BUTTON"
    id=2
  ),
  ActionElement(

```

```

...
content="About"
details={ href: "https://google.com/about" }
relevance={ content: Relevancy.LOW, location: Relevancy.LOW }
type="LINK"
id=3
)
]

```

Prompt You are tasked to predict the next action to achieve the given objective. Imagine you are imitating humans using a web browser to achieve an objective, step by step.

You can take these actions:

CLICK [X] - click element with id X. You can only click on LINK and BUTTON!

TYPE [X] 'text' - type the specified text into INPUT element with id X.

TYPESUBMIT [X] 'text' - same as TYPE, followed by an ENTER key press to submit forms, such as search inputs.

LINK, INPUT, BUTTON elements are represented like this:

```

[1] LINK 'link text' (href='https://example.com')
[2] BUTTON 'button text'
[3] INPUT 'placeholder or label text' (value='initial value')

```

Based on your given objective, issue whatever command you believe will get you closest to achieving your goal.

The following are some examples:

Page URL: <https://gmail.com>

Page title: `Inbox - Gmail`

Actions:

- [1] INPUT 'Search mail'
- [2] BUTTON 'Compose'
- [3] LINK 'Inbox' (href='/inbox')
- [4] LINK 'Sent' (href='/sent')

Objective: Find the value of attribute `date of birth` of entity `Alex`.

Reasoning: **Let's think step by step. We need to find the date of birth of Alex. Since the current page is Gmail inbox, we should start by searching for emails related to Alex. We can search `Alex` in the search input. Therefore, I will issue the command `TYPESUBMIT [1] 'Alex'`.**

Command: TYPESUBMIT [1] 'Alex'

=====

((1 example omitted))

Page URL: Google

Page title: <https://google.com>

Actions:

- [1] INPUT 'Search Google'

- [2] BUTTON 'I'm feeling lucky'
- [3] LINK 'About' (href='/about')

Objective: Find the value of attribute `studied at` of entity `Alex`.

Reasoning: Let's think step by step.

Response Parsing Using the parser from the prompt module, the last string that matches ACTION_TYPE [ID] 'VALUE' is converted into an Action object, including the ActionElement with the matching ID.

4.2 Browser Extension

The browser extension is implemented using Plasmo¹³, a browser extension SDK framework. The extension is composed of 1 extension page, 2 background service workers and 2 content scripts communicating through the messaging API from Plasmo. See Appendix C.3 for the extension manifest.json.

4.2.1 User Interface

The user interface (UI) is implemented as an *extension page*, written in TypeScript and built using React¹⁴, a frontend web development library. Additionally, components from shadcn/ui¹⁵, a collection of styled Radix UI¹⁶ components, were used to create interactive UI components such as accordion, switch and dialogs, and additional UI features such as dark mode and toast messages.

The extension was developed as a side panel to persist throughout webpage navigation. The UI provides interactive elements that such as button to initiate the agent pipeline, dropdown to choose the inference model, switch to enable the continuous mode and button to automatically organize the graph. Figure 11 and Figure 12 below shows the screenshot of the extension UI.

Clicking the Locate button on the UI triggers the process.ts background service worker to start processing the current webpage and start the navigation, shown in code excerpt below.

client/background/messages/process.ts

```
const handler: PlasmoMessaging.MessageHandler<RequestBody, ResponseBody> = async (req, res) => {
  const request = req.body

  // get webpage data from content script
  let webpageData: WebpageData
  try {
    webpageData = await sendToContentScript<any, WebpageData>({
      name: "get-webpage-data"
    })
  } catch (e) {
    console.error(CHECK_EXTENSION, e)
    res.send({ error: CHECK_EXTENSION, isComplete: false })
    return
  }

  // response object to send it to the API server
  const processRequest: ProcessRequestBody = {
    data: webpageData,
    query: request.query,
  }
}
```

¹³ <https://www.plasmo.com/>

¹⁴ <https://react.dev/>

¹⁵ <https://ui.shadcn.com/>

¹⁶ <https://www.radix-ui.com/>

```

    previous_actions: request.previousActions
  }
  let process: Response = null;

  try {
    // send request to `process` endpoint
    process = await fetch(`http://localhost:8000/process?model=${request.model}`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(processRequest)
    })
  } catch (e) {
    console.error(CHECK_NETWORK, e)
    res.send({ error: CHECK_NETWORK, isComplete: false })
  }

  if (!process.ok) {
    // error while processing
    res.send({ error: process.statusText, isComplete: false })
    return
  }

  // response from the API server
  const processResponse: ProcessResponseBody = await process.json()

  // response object to send it back to the extension page
  const response: ResponseBody = {
    results: processResponse.results,
    confidenceLevel: processResponse.confidence_level,
    action: processResponse.next_action ? getActionString(processResponse.next_action) : null,
    isComplete: false
  }

  if (processResponse.next_action === null) {
    // no next action, process is complete
    res.send({ ...response, isComplete: true })
  } else {
    // next action exists, execute action
    const executeActionResponse = await sendToContentScript<
      ExecuteActionRequestBody, ExecuteActionResponseBody
    >({
      name: "execute-action",
      body: {
        action: processResponse.next_action,
        continuous: request.continuous
      }
    })

    // action executed, continue process
    res.send({ ...response, actionResult: executeActionResponse })
  }
}

```

4.2.2 Webpage Encoding

This task is implemented in a *content script*, where the extension can access the currently active tab via the document DOM object. The script encodes the HTML code of the DOM object into a base64 string and sends the webpage data as a response, shown in the code excerpt below.

client/contents/get-webpage-data.ts

```
import { encode } from 'js-base64';

const getWebpageData = (): ResponseBody => {
  return {
    url: document.location.href,
    htmlBase64: encode(document.documentElement.outerHTML),
    imageBase64: "",
    language: document.documentElement.lang && "en"
  }
}

// listen for messages from the background script
listenMessage<any, ResponseBody>(async (req, res) => {
  if (req.name === "get-webpage-data") {
    const result = getWebpageData()
    res.send(result)
  }
})
```

4.2.3 Action Execution

This task is implemented in a *content script*, and uses a *background service worker* to execute actions that require higher privilege. Content script listens to requests from the background service worker process.ts and executes the provided action on the active tab, shown in code excerpt below.

client/contents/execute-action.ts

```
const getElementByXpath = (path: string): HTMLElement => {
  const element = document.evaluate(
    path,
    document,
    null,
    XPathResult.FIRST_ORDERED_NODE_TYPE,
    null
  ).singleNodeValue as HTMLElement

  return element
}

const executeAction = (req: RequestBody): ResponseBody => {
  const action = req.action

  try {
    const element = getElementByXpath(action.element.xpath)

    if (!element) {
      console.error("Element not found", action.element.xpath)
      return { error: `Element not found with xpath=${action.element.xpath}` }
    }

    const actionDescription = getActionDescription(action)
  }
}
```

```

// confirm before executing action if continuous mode is off
if (req.continuous || confirm(`Confirm executing action \`${actionDescription}\`?`)) {
  switch (action.type) {
    case "CLICK":
      clickElement(element)
      break
    case "TYPE":
      typeElement(element as HTMLInputElement, action.value)
      break
    case "TYPESUBMIT":
      typeSubmitElement(element as HTMLInputElement, action.value)
      break
    default:
      console.error("Unsupported action type", action.type)
      return { error: "Unsupported action type" }
  }

  console.info(`Executed action \`${actionDescription}\``)

  return { error: null }
} else {
  console.info(`Canceled action \`${actionDescription}\``)
  return { error: "Action canceled" }
}

} catch (e) {
  console.error("Error while executing action", e)
  return { error: "Error while executing action" }
}
}

```

clickElement

This function uses the DOM object available on the content script, using `element.click()` on the element to simulate a *user click* event.

typeElement

Similar to the `clickElement` function above, this function uses `element.focus()` to focus on the element and updates the value of the `element.value` attribute to simulate a *user input* on an `<input>` element.

typeSubmitElement

In contrast to other function, this action requires higher privilege in order to simulate a *user keypress*. The Google Chrome browser offers a way to simulate a keypress using the `Input.dispatchKeyEvent` method from the Chrome DevTools Protocol¹⁷ API, which is only accessible from the background service worker context. This function sends a message to the `send-command.ts` worker including the list of commands to execute using the `chrome.debugger.sendCommand` API to simulate a `ENTER keypress event`.

4.3.4 Graph Visualization

This task is implemented on the *extension page* using React Flow¹⁸, a React component library for creating node-based interactive diagrams. Additionally, an auto layout feature is implemented using `d3-force`, a physical force simulation library, enabling users to automatically organize graphs with complex node-edge composition.

¹⁷ <https://chromedevtools.github.io/devtools-protocol/>

¹⁸ <https://reactflow.dev/>

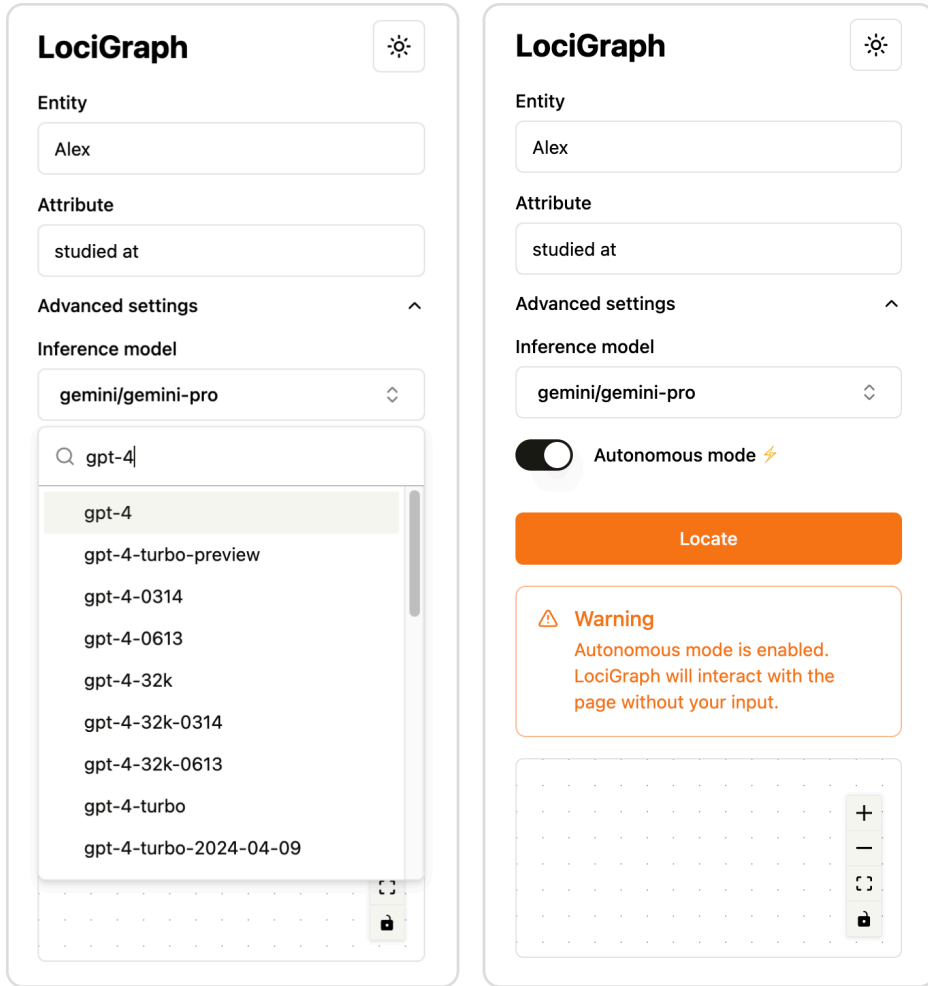


Figure 11: UI components for configuring the inference model and the autonomous mode.

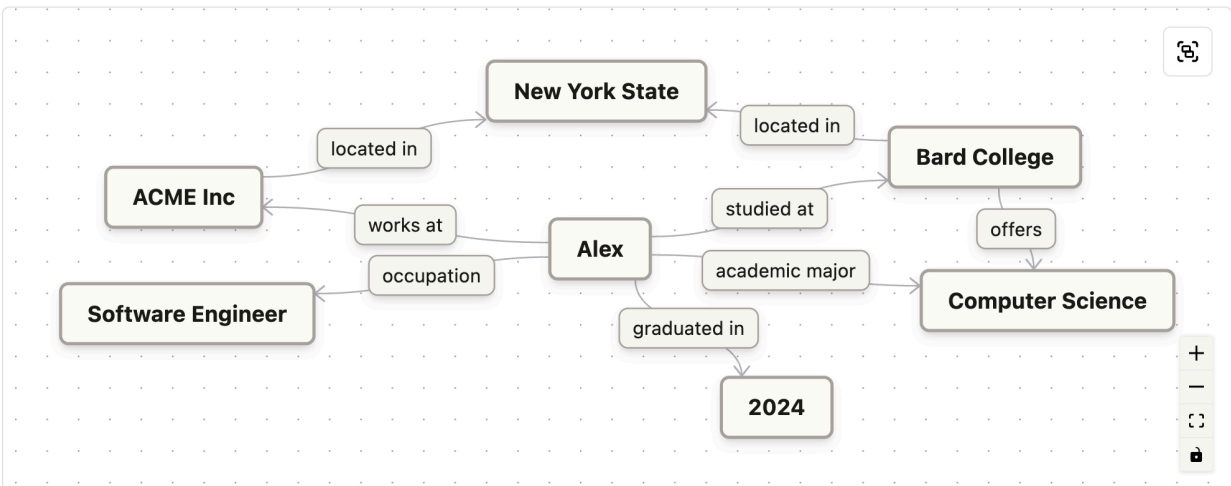


Figure 12: An example of the graph visualization.

5 Experiments

This section reports the results of experiments from a small benchmark (§5.1) and the use case described in Section 3 (§5.2). See Appendix B for the detailed overview and additional episodes.

5.1 Quantitative Assessment: Wikipedia

Benchmark A small dataset of 130 relations from Wikipedia is sampled from the test split of the SRED^{FM} dataset [18]. Within the test split, relations in the English language with 10 attributes¹⁹ are selected and relations with multiple values (e.g. [Google, founded by, Larry Page; Sergey Brin]) are removed in order to simplify the evaluation. From ~2000 relations that satisfy these conditions, 200 relations were randomly sampled and relations with ambiguous entity names (e.g. acronyms) and outdated values (i.e. the Wikipedia article has been updated) were removed, resulting in 130 relations in total. LLM `gemin-1.0-pro` [49] was used for the inference and the navigation was restarted from the Wikipedia landing page²⁰ for each query. Figure 13 below shows example queries.

Query	Target	Results
[WebKit, developer, ?]	Apple	[WebKit, developer, Apple]
[Leonard Sarason, educated at, ?]	Yale University	[Leonard Sarason, educated at, Yale University] [Leonard Sarason, educated at, New York University] [Leonard Sarason, studied, music composition] [Leonard Sarason, studied, Mathematics]
[Bard High School Early College 2011 Newark, inception, ?]		[Bard High School Early College Newark, inception, 2011] [Bard High School Early College Newark, located in, Newark, New Jersey] [Bard High School Early College Newark, school district, Newark Public Schools]

Figure 13: Examples of the benchmark results. See Appendix B.3 for the full list of queries and results.

Results The agent extracted correct relations for 127 queries out of 130. We manually reviewed the results and determined whether it was a satisfactory answer to the query. The agent failed to find a correct value for 3 queries; 2 failed due to the element ranking algorithm not being able to rank the relevant element with the given query, and 1 failed due to the *extract* agent hallucinating results from too few elements. In successful cases, the agent often correctly identified additional relevant relations, resulting in more than half of all results containing 3+ relations. Figure 14 below shows the overview of the results.

Outcome	Count	Details
Success	127	Results contained a relation with a semantically correct target value.
Failure in <i>rank</i>	2	The <i>rank</i> process failed to find the relevant element due to keyword expansion not generating the keyword contained in the element.
Failure in <i>extract</i>	1	The <i>extract</i> agent hallucinated a relation due to lack of content.

Figure 14: Analysis of the results and failure cases.

¹⁹ Attributes used to filter relations: developer, educated at, employer, founded by, headquarters location, inception, occupation, owned by, place of birth, publication date

²⁰ <https://www.wikipedia.org/>

Generalization Capabilities Testing on Wikipedia articles, the evaluation shows that the agent is capable of navigating to the correct webpage using the search functionality. The *act* agent usually issues the TYPESUBMIT action for search inputs, but in a few cases, the agent issues a TYPE action and then issues a following CLICK action on the search button, adding an unnecessary step. Although the *act* agent is only shown 2 simple in-context examples (See Appendix A.2 for prompt templates), the agent is capable of generalizing the example to more complex webpages with hundreds of available actions.

5.2 Qualitative Assessment: Gmail

To better understand the capabilities of the agent framework beyond Wikipedia, the framework is qualitatively tested on the *focus crawling* use case. In this task, the agent starts the navigation from the Gmail inbox²¹ with the following query: [OpenAI, founded on, ?]. The agent is expected to search for the email containing the link to the relevant webpage, open the email, click the link in the email and search for the information from the webpage. The screenshots and LLM responses of this task are described in Appendix B.2.

Generalization Capabilities From 5 test runs, the evaluation shows that the *act* agent is capable of successfully navigating through webpages and finding the correct relation in 4 out of 5 runs. Additionally, the agent consistently recognizes the action LINK [X] {email subject} as an email and opens the email, even though there are no such ICL examples provided.

6 Discussion

6.1 Experimental Findings

The experiments (§5) preliminarily show that LLM agents are generally capable of reasoning with websites represented in textual format, extracting information, and navigating to relevant webpages. On the Wikipedia evaluation, some failure edge cases were identified where the keyword expansion was not comprehensive enough to find the relevant element in the webpage, as shown in Figure 15 below.

Query	Expanded Keywords	Relevant Content
[Suze Rotolo, employer, ?]	['employed', 'organisation', 'worked', ...]	As an artist, she specialized in artists' books and <i>taught</i> at the Parsons School of Design in New York City.
[Garry Hagberg, employer, ?]	['employed', 'organisation', 'worked', ...]	Hagberg <i>became a professor</i> of philosophy at Bard College in 1990. ... At Bard, he <i>teaches</i> specialized courses ...

Figure 15: Analysis of the failure results from the *rank* process. The expanded keywords didn't include the word *teach*, failing to find the relevant content from the webpage. The failure case from the *extract* agent roots from the same problem, where the *rank* process failed to find the relevant content and the *extract* agent hallucinated a nonexistent relation from the content.

These findings confirm the findings from existing research on web navigation agents, that LLMs are capable for simple web navigation tasks if the website is represented in a simplified text format. Additionally, the failure results show using synonyms and aliases might not be comprehensive enough to find contents that are indirectly related to the query.

²¹ <https://mail.google.com/mail/#inbox>

6.2 Limitations

Lack of Comprehensive Tests The tests only contain simple tasks that require 10 or fewer actions to reach the webpage with relevant content, which leaves the evaluation of more complex tasks such as clicking buttons on the webpage to find content or navigating websites that do not have search functionality. Additionally, less than a handful of websites and only 1 LLM model were used in testing, limiting the generalizability of evaluation on a wider range of websites or model-specific behaviors of LLM agents. However, this approach serves as an exploration of a novel method for data mining on the non-public web. Extending or modifying existing web navigation benchmarks for KGC tasks could be one option for building a comprehensive benchmark for future works.

Lack of Context Sharing between Agents In rare cases, the *evaluate* agent rejects the extracted response from the *extract* agent when the relation value seems irrelevant without the context. For example, if the *extract* agent extracts the relation [Alex, founded, Founded] from the content “Alex founded his new company called *Founded*.”, the *evaluate* agent will classify the relation as incorrect, as it seems like a mistake without the context of the original content. Such limitations can be addressed through modifying the prompt generation for *evaluate* agent to include a snippet of original text content.

Lack of Support for Backtracking The current implementation of the *act* agent only supports going forward in navigation, not allowing the agent to go back to the previous page and try a different action. This can limit the number of potentially relevant webpages visited, leading to incomplete queries. Extending the action space to include the action can enable agents to navigate webpages in all directions.

Lack of Support for HTML Elements Although LINK, BUTTON, and INPUT action elements are the most common interactive elements on the web, other interactive elements such as `<select>` element for dropdown menus or complex `<form>` elements are not fully supported by this framework.

6.3 Future Work

This section discusses several ideas to improve the framework for future iterations.

Multimodal Input As shown in the `imageBase64` attribute in the `WebpageData` class, visual data of the webpage can be used for a more comprehensive understanding of the webpage, similar to SeeAct [50], VisualWebArena [51], and Pix2Act [52]. LMMs such as GPT-4V [51], Gemini Vision [47], or Fuyu [52] can be used for reasoning with both image and textual representation.

Improving LLM Agents Instead of using pre-written prompts with in-context learning, utilizing fine-tuning LLM with more examples, multi-agent cooperation, automatic chain-of-thought prompting or top-K verbalized confidence prompting can further improve the accuracy and the robustness of the framework. *Multi-agent KGC* [15] is a method for using multiple agents for the subtasks of KGC. For example, there can be an NER-specific agent and a RE-specific agent working together instead of using a single agent for the entire KGC task. *Automatic chain-of-thought prompting* [54] is a technique where the in-context learning examples are dynamically generated in the prompt. For example, when the query attribute is related to a person, the in-context examples for the *extract* prompt can be generated, including related extraction examples such as “born in” or “works at”, instead of unrelated attributes such as “founded at.” *Top-K verbalized confidence prompting* [55] is a technique of eliciting the LLM to respond with a confidence score of its response, using prompts such as “Provide your k best guesses and the probability that each is correct for the following question.” The confidence score generated by this prompt can be used to filter results to ensure all of the relations are correct or returned with the relations to indicate the trustworthiness of individual results.

Alternative Webpage Representation Methods Instead of using a rule-based algorithm to find relevant elements, other approaches such as using the browser-generated accessibility tree (similar to WebArena [25]) or using pre-trained language models fine-tuned on HTML code such as HTMLM [37], WebN-T5 [56] or WebLINX [57], can be used to build a more robust ranking algorithm that can infer semantic relations from the HTML structure.

Improving Keyword Expansion As shown in the experimental findings (§6.1), using synonyms and aliases for the keyword expansion was not enough for some edge cases. Instead of finding for the exact keyword match in the content, more sophisticated keyword search such as comparing word similarity can be used to handle such edge cases.

Comprehensive KGC The framework currently only completes the RE task within the subtasks of KGC, lacking support for NER and other subtasks. Using the entity type extracted from the mREBEL model or using other transformer-based models such as DREEAM [58] or UniST [59] can add the NER capability to the framework, enabling it to create a more comprehensive KG. Additionally, LLM-based frameworks such as multi-agent cooperation for KGC [15] can be used to extend the KGC capabilities beyond RE and NER.

Improving Query UI Separating the query into *entity* and *attribute* can be unintuitive for non-technical users who are not familiar with the concept of KG. To assist in constructing the query, an autocomplete feature can be added to the *attribute* input using the Wikidata properties (e.g. typing “college” into the attribute field will suggest “studied at” or “graduated in”). Additionally, supporting multiple attributes for in a single query can be useful for advanced users who already know their query. Similar to other generative AI applications, combining the query into a single field of natural language question (e.g. “Where did Alex go to college?” instead of [Alex, studied at, ?]) can further lower the perceived complexity for non-technical users.

Improving User Experience Additional features and tweaks can further improve the user experience. For example, adding an import/export option that integrates with existing KG databases such as Neo4j²² or EdgeDB²³ can significantly improve the result export workflow for users working with KG databases. Additionally, using HTTP 102 requests with status messages to display what the agent pipeline is currently processing (e.g. “Parsing webpage” or “Extracting relations” instead of “Processing”) can improve the perceived performance by showing how the webpage is being processed.

²² <https://neo4j.com/>

²³ <https://www.edgedb.com/>

7 Conclusion

This work introduced LociGraph, an AI agent framework for autonomously navigating and extracting knowledge triplets from non-public webpages. The framework includes the agent pipeline that processes the webpage and the browser extension that interacts with the web browser, implemented with state-of-the-art developments in generative AI. The framework was evaluated with real-world webpages, showing preliminary evidence of LLM agents being capable of completing the query by generalizing the examples provided, but the element ranking algorithm struggles to find indirectly relevant content using the keyword expansion. This work hopes browser-based knowledge graph construction tasks and this framework can further expand the capabilities of web navigation agents.

References

- [1] OpenAI, “Introducing ChatGPT.” [Online]. Available: <https://openai.com/blog/chatgpt>
- [2] L. Wang *et al.*, “A Survey on Large Language Model based Autonomous Agents,” 2023, doi: 10.48550/ARXIV.2308.11432.
- [3] L. Zhong, J. Wu, Q. Li, H. Peng, and X. Wu, “A Comprehensive Survey on Automatic Knowledge Graph Construction,” 2023, doi: 10.48550/ARXIV.2302.05019.
- [4] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, “Unifying Large Language Models and Knowledge Graphs: A Roadmap,” 2023, doi: 10.48550/ARXIV.2306.08302.
- [5] “RDF - Semantic Web Standards.” Accessed: Apr. 16, 2024. [Online]. Available: <https://www.w3.org/RDF/>
- [6] D. B. Lenat, “CYC: a large-scale investment in knowledge infrastructure,” *Commun. ACM*, vol. 38, no. 11, pp. 33–38, Nov. 1995, doi: 10.1145/219717.219745.
- [7] F. M. Suchanek, G. Kasneci, and G. Weikum, “Yago: a core of semantic knowledge,” in *Proceedings of the 16th international conference on World Wide Web*, in WWW ’07. New York, NY, USA: Association for Computing Machinery, May 2007, pp. 697–706. doi: 10.1145/1242572.1242667.
- [8] N. F. Noy *et al.*, “Protégé-2000: An Open-Source Ontology-Development and Knowledge-Acquisition Environment,” *AMIA Annu Symp Proc*, vol. 2003, p. 953, 2003.
- [9] J. R. Finkel, T. Grenager, and C. Manning, “Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling,” in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, K. Knight, H. T. Ng, and K. Oflazer, Eds., Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 363–370. doi: 10.3115/1219840.1219885.
- [10] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, “Open information extraction from the web,” in *Proceedings of the 20th international joint conference on Artificial intelligence*, in IJCAI’07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jan. 2007, pp. 2670–2676.
- [11] R. Angles and C. Gutierrez, “Survey of graph database models,” *ACM Comput. Surv.*, vol. 40, no. 1, p. 1:1-1:39, Feb. 2008, doi: 10.1145/1322432.1322433.
- [12] H. Ye, N. Zhang, H. Chen, and H. Chen, “Generative Knowledge Graph Construction: A Review.” arXiv, Sep. 18, 2023. doi: 10.48550/arXiv.2210.12714.
- [13] Y. Zhu *et al.*, “LLMs for Knowledge Graph Construction and Reasoning: Recent Capabilities and Future Opportunities.” arXiv, Feb. 22, 2024. doi: 10.48550/arXiv.2305.13168.
- [14] Y. Zhang, Z. Chen, L. Guo, Y. Xu, W. Zhang, and H. Chen, “Making Large Language Models Perform Better in Knowledge Graph Completion.” arXiv, Apr. 14, 2024. doi: 10.48550/arXiv.2310.06671.
- [15] H. Ye, H. Gui, A. Zhang, T. Liu, W. Hua, and W. Jia, “Beyond Isolation: Multi-Agent Synergy for Improving Knowledge Graph Construction.” arXiv, Dec. 29, 2023. doi: 10.48550/arXiv.2312.03022.
- [16] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, “ERNIE: Enhanced Language Representation with Informative Entities,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Márquez, Eds., Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 1441–1451. doi: 10.18653/v1/P19-1139.
- [17] M. E. Peters *et al.*, “Knowledge Enhanced Contextual Word Representations,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 43–54. doi: 10.18653/v1/D19-1005.
- [18] P.-L. Huguet Cabot, S. Tedeschi, A.-C. Ngonga Ngomo, and R. Navigli, “RED^{FM}: a Filtered and Multilingual Relation Extraction Dataset,” 2023, doi: 10.48550/ARXIV.2306.09802.
- [19] Y. Tang *et al.*, “Multilingual Translation with Extensible Multilingual Pretraining and Finetuning.” arXiv, Aug. 02, 2020. doi: 10.48550/arXiv.2008.00401.
- [20] R. Nakano *et al.*, “WebGPT: Browser-assisted question-answering with human feedback.” arXiv, Jun. 01, 2022. doi: 10.48550/arXiv.2112.09332.
- [21] S. Yao, H. Chen, J. Yang, and K. Narasimhan, “WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents.” arXiv, Feb. 07, 2023. doi: 10.48550/arXiv.2207.01206.
- [22] MDN Contributors, “Document Object Model (DOM) - Web APIs | MDN,” mdn web docs. Accessed: May

- 01, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- [23] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang, “World of Bits: An Open-Domain Platform for Web-Based Agents,” in *Proceedings of the 34th International Conference on Machine Learning*, PMLR, Jul. 2017, pp. 3135–3144. Accessed: Apr. 28, 2024. [Online]. Available: <https://proceedings.mlr.press/v70/shi17a.html>
- [24] X. Deng *et al.*, “Mind2Web: Towards a Generalist Agent for the Web,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., Curran Associates, Inc., 2023, pp. 28091–28114. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf
- [25] S. Zhou *et al.*, “WebArena: A Realistic Web Environment for Building Autonomous Agents,” 2023, doi: 10.48550/ARXIV.2307.13854.
- [26] Adept Team, “ACT-1: Transformer for Actions,” Adept AI. Accessed: Apr. 28, 2024. [Online]. Available: <https://www.adept.ai/blog/act-1>
- [27] MultiOn, “Announcing MultiOn: Building a Brighter Future for Humanity with AI Agents — MultiOn AI,” MultiOn. Accessed: Apr. 28, 2024. [Online]. Available: <https://www.multion.ai/blog/multion-building-a-brighter-future-for-humanity-with-ai-agents>
- [28] HyperWrite, “AI Personal Assistant | HyperWrite AI Agent.” Accessed: Apr. 28, 2024. [Online]. Available: <https://www.hyperwriteai.com/personal-assistant>
- [29] R. Nogueira and K. Cho, “End-to-End Goal-Driven Web Navigation,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2016. Accessed: Apr. 29, 2024. [Online]. Available: <https://proceedings.neurips.cc/paper/2016/hash/1579779b98ce9edb98dd85606f2c119d-Abstract.html>
- [30] S. Chakrabarti, “Focused Web Crawling,” in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds., Boston, MA: Springer US, 2009, pp. 1147–1155. doi: 10.1007/978-0-387-39940-9_165.
- [31] R. Assouel *et al.*, “The Unsolved Challenges of LLMs as Generalist Web Agents: A Case Study,” in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. Accessed: Apr. 29, 2024. [Online]. Available: <https://openreview.net/forum?id=jt3il4fC5B>
- [32] Y. Zhou, Y. Sheng, N. Vo, N. Edmonds, and S. Tata, “Simplified DOM Trees for Transferable Attribute Extraction from the Web.” arXiv, Jan. 07, 2021. doi: 10.48550/arXiv.2101.02415.
- [33] Q. Hao, R. Cai, Y. Pang, and L. Zhang, “From one tree to a forest: a unified solution for structured web data extraction,” in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, Beijing China: ACM, Jul. 2011, pp. 775–784. doi: 10.1145/2009916.2010020.
- [34] B. Y. Lin, Y. Sheng, N. Vo, and S. Tata, “FreeDOM: A Transferable Neural Architecture for Structured Information Extraction on Web Documents,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Virtual Event CA USA: ACM, Aug. 2020, pp. 1092–1102. doi: 10.1145/3394486.3403153.
- [35] Q. Wang, Y. Fang, A. Ravula, F. Feng, X. Quan, and D. Liu, “WebFormer: The Web-page Transformer for Structure Information Extraction,” in *Proceedings of the ACM Web Conference 2022*, Virtual Event, Lyon France: ACM, Apr. 2022, pp. 3124–3133. doi: 10.1145/3485447.3512032.
- [36] Y. Xu *et al.*, “LayoutLMv2: Multi-modal Pre-training for Visually-Rich Document Understanding.” arXiv, Jan. 09, 2022. Accessed: Apr. 29, 2024. [Online]. Available: <http://arxiv.org/abs/2012.14740>
- [37] A. Aghajanyan *et al.*, “HTLM: Hyper-Text Pre-Training and Prompting of Language Models.” arXiv, Jul. 14, 2021. Accessed: Apr. 29, 2024. [Online]. Available: <http://arxiv.org/abs/2107.06955>
- [38] X. Deng, P. Shiralkar, C. Lockard, B. Huang, and H. Sun, “DOM-LM: Learning Generalizable Representations for HTML Documents.” arXiv, Jan. 25, 2022. Accessed: Apr. 29, 2024. [Online]. Available: <http://arxiv.org/abs/2201.10608>
- [39] J. Li, Y. Xu, L. Cui, and F. Wei, “MarkupLM: Pre-training of Text and Markup Language for Visually-rich Document Understanding.” arXiv, Mar. 11, 2022. Accessed: Apr. 29, 2024. [Online]. Available: <http://arxiv.org/abs/2110.08518>
- [40] S. Mazumder and O. Riva, “FLIN: A Flexible Natural Language Interface for Web Navigation.” arXiv, Apr. 13, 2021. doi: 10.48550/arXiv.2010.12844.
- [41] N. Friedman, “nat/natbot: Drive a browser with GPT-3,” GitHub. Accessed: Apr. 29, 2024. [Online]. Available: <https://github.com/nat/natbot>

- [42] MDN Contributors, “XPath | MDN,” mdn web docs. Accessed: Apr. 29, 2024. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/XPath>
- [43] MDN Contributors, “ARIA: landmark role - Accessibility | MDN,” mdn web docs. Accessed: Apr. 29, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/roles/landmark_role
- [44] “Wikidata:Statistics - Wikidata.” Accessed: Apr. 29, 2024. [Online]. Available: <https://www.wikidata.org/wiki/Wikidata:Statistics>
- [45] “Wikidata Datamodel Statements - Wikidata - Dashboards - Grafana.” Accessed: Apr. 29, 2024. [Online]. Available: <https://grafana.wikimedia.org/d/000000175/wikidata-datamodel-statements?orgId=1>
- [46] “Open Wordnet Documentation.” Accessed: Apr. 29, 2024. [Online]. Available: <https://globalwordnet.github.io/gwadoc/>
- [47] T. B. Brown *et al.*, “Language Models are Few-Shot Learners.” arXiv, Jul. 22, 2020. doi: 10.48550/arXiv.2005.14165.
- [48] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.” arXiv, Jan. 10, 2023. doi: 10.48550/arXiv.2201.11903.
- [49] Gemini Team *et al.*, “Gemini: A Family of Highly Capable Multimodal Models.” arXiv, Apr. 02, 2024. doi: 10.48550/arXiv.2312.11805.
- [50] OpenAI, “GPT-4V(ision) system card.” Accessed: Apr. 30, 2024. [Online]. Available: <https://openai.com/research/gpt-4v-system-card>
- [51] J. Y. Koh *et al.*, “VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks.” arXiv, Jan. 24, 2024. doi: 10.48550/arXiv.2401.13649.
- [52] P. Shaw *et al.*, “From Pixels to UI Actions: Learning to Follow Instructions via Graphical User Interfaces.” arXiv, Dec. 06, 2023. doi: 10.48550/arXiv.2306.00245.
- [53] R. Bavishi *et al.*, “Fuyu-8B: A Multimodal Architecture for AI Agents.” Accessed: Apr. 30, 2024. [Online]. Available: <https://www.adept.ai/blog/fuyu-8b/>
- [54] Z. Zhang, A. Zhang, M. Li, and A. Smola, “Automatic Chain of Thought Prompting in Large Language Models.” arXiv, Oct. 07, 2022. doi: 10.48550/arXiv.2210.03493.
- [55] M. Xiong *et al.*, “Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs.” arXiv, Mar. 17, 2024. doi: 10.48550/arXiv.2306.13063.
- [56] I. Gur *et al.*, “Understanding HTML with Large Language Models.” arXiv, May 19, 2023. doi: 10.48550/arXiv.2210.03945.
- [57] X. H. Lù, Z. Kasner, and S. Reddy, “WebLINX: Real-World Website Navigation with Multi-Turn Dialogue.” arXiv, Feb. 08, 2024. Accessed: Apr. 29, 2024. [Online]. Available: <http://arxiv.org/abs/2402.05930>
- [58] Y. Ma, A. Wang, and N. Okazaki, “DREEAM: Guiding Attention with Evidence for Improving Document-Level Relation Extraction.” arXiv, Feb. 16, 2023. doi: 10.48550/arXiv.2302.08675.
- [59] J. Y. Huang, B. Li, J. Xu, and M. Chen, “Unified Semantic Typing with Meaningful Label Inference.” arXiv, May 03, 2022. doi: 10.48550/arXiv.2205.01826.

Appendix

The supplementary content includes the following sections:

- **Section A: Implementation Details** Project structure and LLM prompt templates.
- **Section B: Experiment Details** Experiment environment, an example episode, and the benchmark.
- **Section C: Source Code** The source code and the source directory structure.

All artifacts are open-sourced and publicly available online:

Artifact	Link	License
Homepage	https://ntcho.github.io/LociGraph	MIT License
Code Repository	https://github.com/ntcho/LociGraph	MIT License

A Implementation Details

A.1 Project Structure

The project consists of two applications: agent pipeline as the API server and the browser extension. The agent pipeline is built using Python 3.12 with Litestar 2.6.0, and the browser extension is built using TypeScript 5.4.5 with Plasmio 0.86.0 and React 18.3.0. Figure 16 below lists the external packages used and their purpose. A full list of packages can be found in `server/requirements.txt` for the agent pipeline (Appendix C.2) and `client/package.json` for the browser extension (Appendix C.3).

Package	Version	Purpose
requirements.txt		
litestar	2.6.0	API server implementation
lxml	5.1.0	HTML parsing & DOM tree modification implementation
litellm	1.35.17	LLM API integration
tinycss2	1.2.1	CSS parsing implementation
transformers	4.40.0	mREBEL model integration
loguru	0.7.2	Customized logging configuration
package.json		
plasmio	0.86.0	Browser extension implementation
@plasmohq/messaging	^0.6.2	Browser messaging implementation
react	18.3.0	UI component implementation
tailwindcss	^3.4.1	UI component styling
lucide-react	^0.373.0	UI icon import
reactflow	^11.11.2	Interactive graph visualization implementation
@radix-ui	multiple	Interactive UI component implementation
js-base64	^3.7.7	Base64 encode/decode implementation
export-to-csv	^1.2.4	CSV export download implementation

Figure 16: List of external packages used for implementation.

A.2 Prompt Templates

The following are the prompt templates for each LLM agent. Prompt generation and response parsing implementation in `prompt.py` in Appendix C.2.

extract

You are tasked to extract all relation triplets from a webpage. Imagine you are imitating humans using a web browser to find information, step by step.

The following are some examples:

Title: `About Alex | alex.com`

Content:

...

Born

January 1, 2000

New York City, NY, US

...

Query: [Alex, date of birth, ?]

Reasoning: Let's think step by step. We need to find the date of birth of Alex. The content provides the birthday January 1st, 2000. Since the content provides the date of birth, we should extract the relation [Alex, date of birth, January 1st, 2000]. Additionally, the content provides the birthplace New York City. Since this is related to Alex, we should extract the relation [Alex, place of birth, New York City] as an additional relation.

Output:

Query relations:

- [Alex, date of birth, January 1, 2000]

Additional relations:

- [Alex, place of birth, New York City]

=====

Title: `Alex | LinkedIn`

Content:

...

Education

Bard College

Computer Science

Timbuktu High School

Mathematics

...

Query: [Alex, educated at, ?]

Reasoning: Let's think step by step. We need to find the institution where Alex was educated at. The content provides that Alex studied Computer Science at Bard College. Since this is a factual relation and related to query, we should extract the relation [Alex, educated at, Bard College] and [Bard College, studied, Computer Science]. Additionally, the content provides that Bard College offers Computer Science. Since this is a factual relation, we should extract the relation [Bard College, offers, Computer Science] as an additional relation. The content also provides that Alex studied Mathematics at Timbuktu High School. Since this is a factual relation and related to query, we should extract the relation [Alex, educated at, Timbuktu High School] and [Alex, studied, Mathematics].

```
Output:
Query relations:
- [Alex, educated at, Bard College]
- [Alex, educated at, Timbuktu High School]
Additional relations:
- [Alex, studied, Computer Science]
- [Bard College, offers, Computer Science]
- [Alex, studied, Mathematics]
=====

Title: `Alex - Wikipedia`
Content:
...
Alex works at ACME Inc as a software engineer. Previously, Alex worked at XYZ Corp as a data
scientist. Alex has a degree in Computer Science from Bard College.
...

Query: [Alex, ?, ?]
Reasoning: Let's think step by step. We need to find all relevant relations for Alex. The
content provides that Alex works at ACME Inc as a software engineer. Since this is related to
Alex, we should extract the relation [Alex, works at, ACME Inc] and [Alex, job title, software
engineer]. The content also provides that Alex previously worked at XYZ Corp as a data
scientist. Since this is related to Alex, we should extract the relation [Alex, worked at, XYZ
Corp] and [Alex, job title, data scientist]. The content also provides that Alex has a degree
in Computer Science from Bard College. Since this is related to Alex, we should extract the
relation [Alex, majored in, Computer Science] and [Alex, graduated from, Bard College].
Output:
Query relations:
- [Alex, works at, ACME Inc]
- [Alex, job title, software engineer]
- [Alex, worked at, XYZ Corp]
- [Alex, job title, data scientist]
- [Alex, majored in, Computer Science]
- [Alex, graduated from, Bard College]
=====

Title: `Fwd: RSVP for Alex's Birthday Party | Gmail`
Content:
...
Hi everyone,

Please RSVP for Alex's birthday party this Friday at 7:00 PM.

Thanks!
...

Query: [Alex, date of birth, ?]
Reasoning: Let's think step by step. We need to find the date of birth of Alex. The content
provides that Alex's birthday party is this Friday at 7:00 PM. Since this content doesn't
provide the date of birth, we should write `No relations found`.
Output:
No relations found
=====

Page title: <title>
Content:
...
<content>
...

Query: <query>
Reasoning: Let's think step by step.
```

evaluate

You are tasked to evaluate relation extraction results for the given query. Imagine you are imitating humans using a web browser to achieve an objective, step by step.

The following are some examples:

Query: [Alex, date of birth, ?]

Extraction results:

- [Alex, birthday, January 1, 2000]
- [Alex, born on, 2000]
- [Alex, born in, New York]

Reasoning: Let's think step by step. We need to find the date of birth of Alex. [Alex, birthday, January 1st, 2000] provides the date of birth as `January 1st, 2000`, therefore this is correct. [Alex, born on, 2000] only provides the year compared to the previous relation, therefore this is incorrect. [Alex, born in, New York] is not related to date of birth, but it is related to the query, therefore this is relevant. Since at least one extraction result is correct, we should `STOP`.

Answer: STOP

Correct relation:

- [Alex, date of birth, January 1, 2000]

Relevant relation:

- [Alex, place of birth, New York]

=====

Query: [Alex, educated at, ?]

Extraction results:

- [Alex, studied, Computer Science]
- [Alex, graduated in, 2020]

Reasoning: Let's think step by step. We need to find the institution where Alex was educated at. [Alex, studied, Computer Science] only provides the academic major, but it is related to the query, therefore this is relevant. [Alex, graduated in, 2020] only provides the year of graduation, but it is related to the query, therefore this is relevant. Since none of the extraction results are correct, we should `CONTINUE`.

Answer: CONTINUE

Relevant relations:

- [Alex, studied, Computer Science]
- [Alex, graduated in, 2020]

=====

Query: [Alex, ?, ?]

Extraction results:

- [Alex, works at, ACME Inc]
- [ACME Inc, location, New York]
- [Alex, job title, software engineer]
- [Foo Corp, headquarters, San Francisco]

Reasoning: Let's think step by step. We need to find all relevant relations for Alex. [Alex, works at, ACME Inc] provides that Alex works at `ACME Inc`, therefore this is correct. [ACME Inc, location, New York] only provides the location of the company, but it is related to the previous relations, therefore this is relevant. [Alex, job title, software engineer] provides the job title of Alex, therefore this is correct. [Foo Corp, headquarters, San Francisco] is not related to the query nor any relations, therefore this is incorrect. Since at least one extraction result is correct, we should `STOP`.

Answer: STOP

Correct relations:

- [Alex, works at, ACME Inc]
- [Alex, job title, software engineer]

Relevant relations:

- [ACME Inc, location, New York]

=====

Query: <query>
Extraction results:
<relations>
Reasoning: Let's think step by step.

act

You are tasked to predict the next action to achieve the given objective. Imagine you are imitating humans using a web browser to achieve an objective, step by step.

You can take these actions:

CLICK [X] - click element with id X. You can only click on LINK and BUTTON!
TYPE [X] 'text' - type the specified text into INPUT element with id X.
TYPESUBMIT [X] 'text' - same as TYPE, except this command presses ENTER to submit the form, such as a search input.

LINK, INPUT, BUTTON elements are represented like this:

```
[1] LINK 'link text' (href='https://example.com')
[2] BUTTON 'button text'
[3] INPUT 'placeholder or label text' (value='initial value')
```

Based on your given objective, issue whatever command you believe will get you closest to achieving your goal.

The following are some examples:

Page URL: <https://gmail.com>

Page title: `Inbox - Gmail`

Actions:

- [1] BUTTON 'Compose'
- [2] INPUT 'Search mail'
- [3] LINK 'Inbox' (href='/inbox')
- [4] LINK 'Sent' (href='/sent')

Objective: Find the value of attribute `date of birth` of entity `Alex`.

Reasoning: Let's think step by step. We need to find the date of birth of Alex. Since the current page is Gmail inbox, we should start by searching for emails related to Alex. We can search with the query `Alex` by typing it in the search mail input. Therefore, I will issue the command `TYPE [2] 'Alex'`.

Command: TYPESUBMIT [2] 'Alex'

=====

Page URL: <https://wikipedia.org/en/Alex>

Page title: `Alex - Wikipedia`

Actions:

- [1] LINK 'Bard College - Alex' (href='bard.edu/people/alex')
- [2] LINK 'Alex | LinkedIn' (href='linkedin.com/in/alex')
- [3] LINK 'ACME Inc - Alex' (href='acme.com/people/alex')

Objective: Find the value of attribute `graduated in` of entity `Alex`.

Reasoning: Let's think step by step. We need to find the year Alex graduated. Since the current page is Alex's Wikipedia page, we should click on the LinkedIn link to find the educational background of Alex. Therefore, I will issue the command `CLICK [2]`.

Command: CLICK [2]

=====

Page URL: <url>
Page title: <title>
Actions:
<actions>
Objective: <objective>
Previous actions:
<previous_actions>
Reasoning: Let's think step by step.

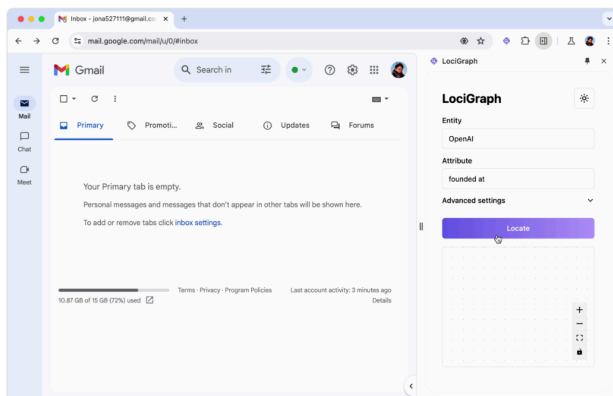
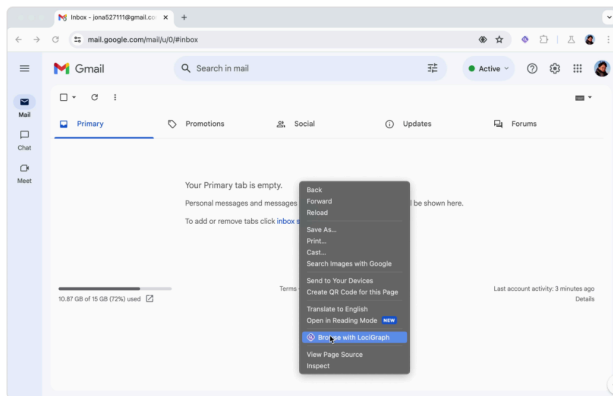
B Experiment Details

B.1 Experiment Environment

All experiments in Section 5 were conducted on a MacBook Pro (2021; Apple M1 Pro with 16GB RAM), with webpages accessed between March 1 and April 30, 2024. Inference model gemini-1.0-pro hosted on Gemini API²⁴ was used for all testing. Additionally, llama-2-70b-chat-hf hosted on Together AI²⁵ was used in the development phase.

B.2 Example Episode

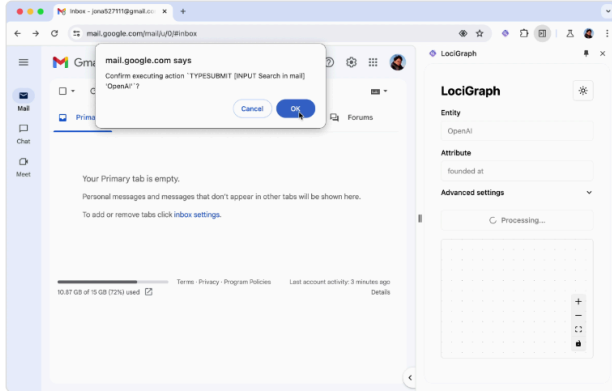
The following are screenshots and LLM responses of the steps taken for the task described in Section 5.2.



((No elements found with rank))

²⁴ <https://ai.google.dev/>

²⁵ <https://www.together.ai/>

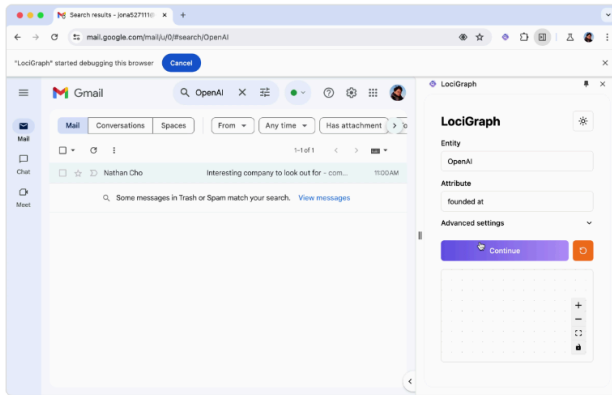


You are tasked to predict the next action to achieve the given objective. Imagine you are imitating humans using a web browser to achieve an objective, step by step. ((detailed instruction & 2 ICL examples hidden))

Page URL: <https://mail.google.com/mail/u/0/#inbox>
 Page title: Inbox - jona527111@gmail.com - Gmail

Actions:
 - [1] INPUT 'Search in mail'
 - [2] LINK 'Primary'
 - [3] LINK 'Promotions'
 - [4] LINK 'Social'
 - [5] LINK 'Updates'
 - [6] LINK 'Forums'
 ((21 more actions hidden))

Objective: Find the value of attribute 'founded at' of entity 'OpenAI'.
 Reasoning: Let's think step by step. We need to find when OpenAI was founded. Since the current page is the Gmail inbox, we should start by searching for emails related to OpenAI. We can search 'OpenAI' in the search input. Therefore, I will issue the command 'TYPESUBMIT [2] 'OpenAI''.
 Command: TYPESUBMIT [2] 'OpenAI'

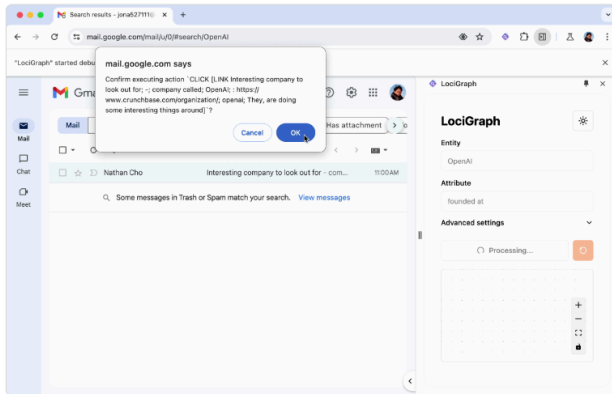


You are tasked to extract all relation triplets from a webpage. Imagine you are imitating humans using a web browser to find information, step by step. ((3 ICL examples hidden))

Page title: Search results - jona527111@gmail.com - Gmail
 Content:

[table]
 Nathan Cho | Interesting company to look out for; - company called OpenAI : <https://www.crunchbase.com/organization/openai> They, are doing some interesting things around | 11:00 AM

Query: [OpenAI, founded at, ?]
 Reasoning: Let's think step by step. The content doesn't provide sufficient information to find the founding date of OpenAI. Therefore, we should write 'No relations found'.
 Output:
 No relations found

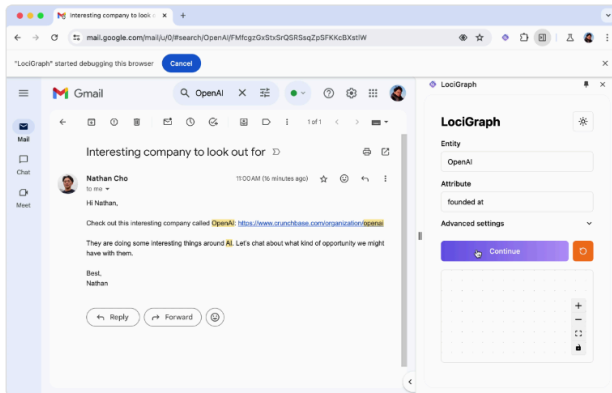


You are tasked to predict the next action to achieve the given objective. Imagine you are imitating humans using a web browser to achieve an objective, step by step. ((detailed instruction & 2 ICL examples hidden))

Page URL: <https://mail.google.com/mail/u/0/#search/OpenAI>
 Page title: Search results - jona527111@gmail.com - Gmail

Actions:
 - [1] INPUT 'Search in mail'
 - [2] BUTTON 'View messages'
 - [3] BUTTON '1; -; ; of; 1'
 - [4] LINK 'Interesting company to look out for; -; company called; OpenAI; : <https://www.crunchbase.com/organization/openai>; They, are doing some interesting things around'
 - [5] INPUT
 - [6] LINK 'Skip to content'
 - [7] LINK 'Using Gmail with screen readers'
 - [8] LINK 'Keyboard shortcuts'
 - [9] LINK 'Gmail (standard view) guide' (href='support.google.com/mail/answer/90559?..')
 - [10] BUTTON 'Compose'
 ((28 more actions hidden))

Objective: Find the value of attribute 'founded at' of entity 'OpenAI'.
 Previous actions:
 - TYPESUBMIT 'OpenAI' (INPUT 'Search in mail')
 Reasoning: Let's think step by step. We need to find the year when OpenAI was founded. Since the current page lists search results of emails containing the keyword 'OpenAI', we should click on the first link to read the email. Therefore, I will issue the command 'CLICK [4]'.
 Command: CLICK [4]

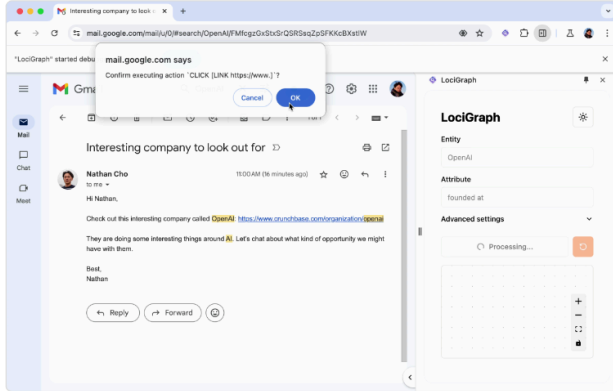


You are tasked to extract all relation triplets from a webpage. Imagine you are imitating humans using a web browser to find information, step by step. ((3 ICL examples hidden))

Page title: Interesting company to look out for - jona527111@gmail.com - Gmail
 Content:

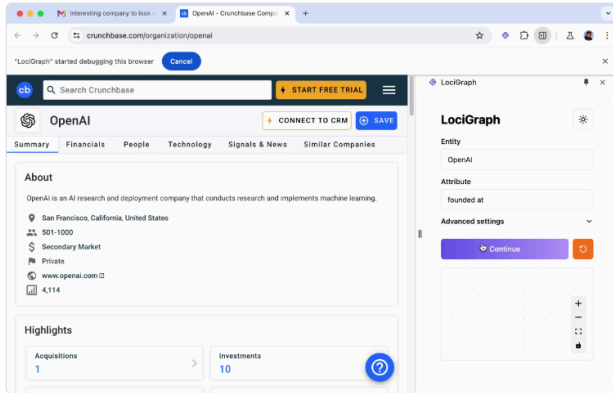
Check out this interesting company called OpenAI : <https://www.crunchbase.com/organization/openai>

Query: [OpenAI, founded at, ?]
 Reasoning: Let's think step by step. We need to find the founded date of OpenAI. The content doesn't provide the founded date, we should write 'No relations found'.
 Output:
 No relations found



You are tasked to predict the next action to achieve the given objective. Imagine you are imitating humans using a web browser to achieve an objective, step by step. ((detailed instruction & 2 ICL examples hidden))

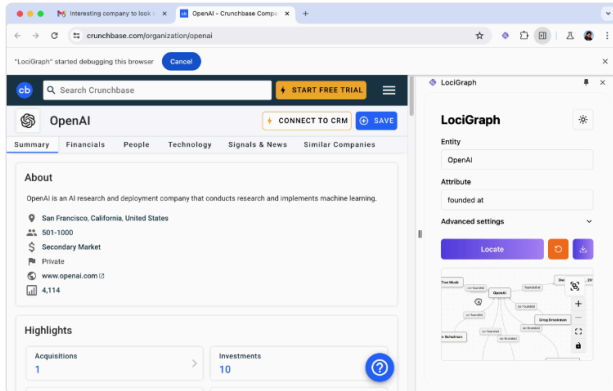
Page URL: <https://mail.google.com/mail/u/0/#search/OpenAI/FMfcg2GxStxSrQSRsqZpSFkKcBxtlW>
 Page title: Interesting company to look out for - jona527111@gmail.com - Gmail
 Actions:
 - [1] INPUT 'Search in mail'
 - [2] LINK 'Nathan Cho; 11:00 AM (16 minutes ago); to; me'
 - [3] LINK 'https://www.' (href= crunchbase.com/organization/openai')
 - [4] LINK 'Reply'
 - [5] LINK 'Forward'
 ((19 more actions hidden))
 Objective: Find the value of attribute 'founded at' of entity 'OpenAI'.
 Previous actions:
 - TYPESUBMIT 'OpenAI' (INPUT 'Search in mail')
 - CLICK (LINK 'Interesting company to look out for; -; company called; OpenAI; : https://www.crunchbase.com/organization/; openai; They, are doing some interesting things around')
 Reasoning: Let's think step by step. We need to find the founding date of OpenAI. Since the current page is a search result page in Gmail, we should click on the OpenAI link to find more information about the company. Therefore, I will issue the command 'CLICK [3]'.
 Command: CLICK [3]



You are tasked to extract all relation triplets from a webpage. Imagine you are imitating humans using a web browser to find information, step by step. ((3 ICL examples hidden))

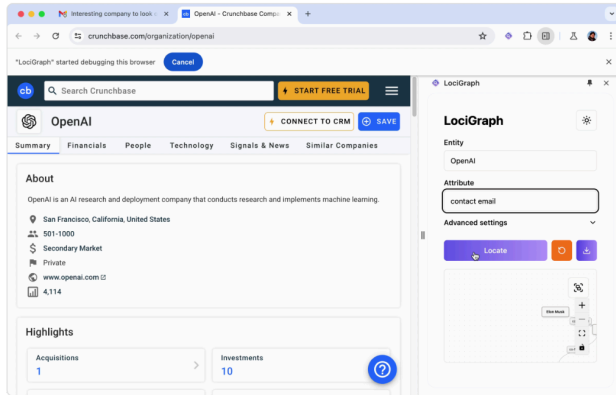
Page title: OpenAI - Crunchbase Company Profile & Funding
 Content:
 ((87 lines of content hidden))
 - Industries
 - Artificial Intelligence (AI)
 - Generative AI
 - Machine Learning
 - Natural Language Processing
 - Software
 - Diversity Spotlight (US Headquarters Only)
 - LGBTQ+ Founded
 - LGBTQ+ Led
 - Headquarters Regions
 - San Francisco Bay Area , West Coast , Western US
 - Founded Date
 - Dec 11, 2015
 - Founders
 - Elon Musk , Greg Brockman , Ilya Sutskever , John Schulman , Sam Altman , Wojciech Zaremba
 - Operating Status
 - Active
 - Last Funding Type
 - Secondary Market
 - Also Known As
 - OpenAI LP
 - Legal Name
 - OpenAI Inc.
 - Related Hubs
 - OpenAI Alumni Founded Companies
 - Hub Tags
 - Unicorn
 ((59 lines of content hidden))

Query: [OpenAI, founded at, ?]
 Reasoning: Let's think step by step. We need to find the date when OpenAI was founded. The content provides that OpenAI was founded on December 11th, 2015. Since this is related to OpenAI, we should extract the relation [OpenAI, founded at, December 11th, 2015]. Additionally, the content provides that Elon Musk, Greg Brockman, Ilya Sutskever, John Schulman, Sam Altman, and Wojciech Zaremba are founders of OpenAI. Since this is related to OpenAI, we should extract the relation [Elon Musk, co-founded, OpenAI], [Greg Brockman, co-founded, OpenAI], [Ilya Sutskever, co-founded, OpenAI], [John Schulman, co-founded, OpenAI], [Sam Altman, co-founded, OpenAI], and [Wojciech Zaremba, co-founded, OpenAI].
 Output:
 Query relations:
 - [OpenAI, founded at, December 11th, 2015]
 Additional relations:
 - [Elon Musk, co-founded, OpenAI]
 - [Greg Brockman, co-founded, OpenAI]
 - [Ilya Sutskever, co-founded, OpenAI]
 - [John Schulman, co-founded, OpenAI]
 - [Sam Altman, co-founded, OpenAI]
 - [Wojciech Zaremba, co-founded, OpenAI]



You are tasked to evaluate relation extraction results for the given query. Imagine you are imitating humans using a web browser to achieve an objective, step by step. ((3 ICL examples hidden))

Query: [OpenAI, founded at, ?]
 Extraction results:
 - [OpenAI, founded at, December 11th, 2015]
 - [Elon Musk, co-founded, OpenAI]
 - [Greg Brockman, co-founded, OpenAI]
 - [Ilya Sutskever, co-founded, OpenAI]
 - [John Schulman, co-founded, OpenAI]
 - [Sam Altman, co-founded, OpenAI]
 - [Wojciech Zaremba, co-founded, OpenAI]
 Reasoning: Let's think step by step. We need to find when OpenAI was founded. Therefore this is correct. [Elon Musk, co-founded, OpenAI] only provides that Elon Musk was one of the co-founders of OpenAI, but it is related to the query, therefore this is relevant. [Greg Brockman, co-founded, OpenAI], [Ilya Sutskever, co-founded, OpenAI], [John Schulman, co-founded, OpenAI], [Sam Altman, co-founded, OpenAI], and [Wojciech Zaremba, co-founded, OpenAI] also provide information on who co-founded OpenAI, but it is related to the query, therefore this is relevant. Since at least one extraction result is correct, we should 'STOP'.
 Answer: STOP
 Correct relation:
 - [OpenAI, founded at, December 11th, 2015]
 Relevant relations:
 - [Elon Musk, co-founded, OpenAI]
 - [Greg Brockman, co-founded, OpenAI]
 - [Ilya Sutskever, co-founded, OpenAI]
 - [John Schulman, co-founded, OpenAI]
 - [Sam Altman, co-founded, OpenAI]
 - [Wojciech Zaremba, co-founded, OpenAI]



You are tasked to extract all relation triplets from a webpage. Imagine you are imitating humans using a web browser to find information, step by step. ((3 ICL examples hidden))

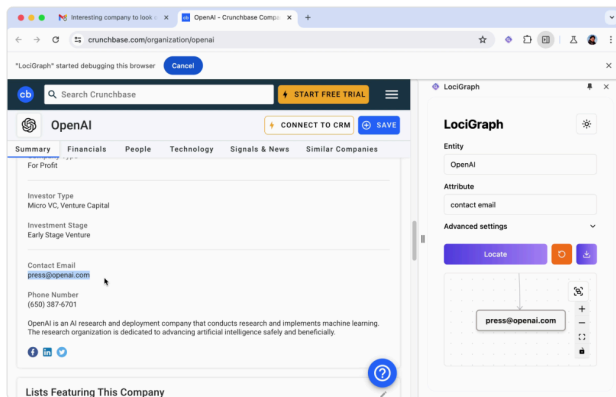
Page title: OpenAI - Crunchbase Company Profile & Funding Content:

((114 lines of content hidden))

- Contact Email press@openai.com
- Phone Number (650) 387-6701

((55 lines of content hidden))

Query: [OpenAI, founded at, ?]
Reasoning: Let's think step by step. We need to find the contact email of OpenAI. The content provides the contact email press@openai.com. Since this is related to OpenAI, we should extract the relation [OpenAI, contact email, press@openai.com].
Output:
Query relations:
- [OpenAI, contact email, press@openai.com]



You are tasked to evaluate relation extraction results for the given query. Imagine you are imitating humans using a web browser to achieve an objective, step by step. ((3 ICL examples hidden))

Query: [OpenAI, contact email, ?]

Extraction results:

- [OpenAI, contact email, press@openai.com]
Reasoning: Let's think step by step. We need to find the contact email of OpenAI. [OpenAI, contact email, press@openai.com] provides the contact email as press@openai.com, therefore this is correct. Since at least one extraction result is correct, we should 'STOP'.
Answer: STOP

Correct relations:
- [OpenAI, contact email, press@openai.com]

B.3 Benchmark Results

The following is the list of queries with target, outcome, and result relations from the Wikipedia benchmark. Outcome symbol ✓ denotes the results contained an exact match, ☑ denotes a match after a manual review, and ✗ denotes no match.

Query with Target	Out. Results
[Garry Hagberg, employer, Bard College]	✗ [Garry Hagberg, employer, ?]
[Suze Rotolo, employer, Parsons School of Design]	✗ [Suze Rotolo, employer, Congress of Racial Equality] [Suze Rotolo, employer, SANE] [Suze Rotolo, member of political party, Congress of Racial Equality] [Suze Rotolo, sibling, Carla] [Suze Rotolo, member of, SANE]
[KIPR, owned by, Cumulus Media]	✗ [KIPR, owned by, Delta Media Corporation]
[3D Construction Kit, developer, Incentive Software]	✓ [3D Construction Kit, developer, Incentive Software] [3D Construction Kit, publication date, 1991] [3D Construction Kit, instance of, Video game development software] [3D Construction Kit, platform, Amiga] [3D Construction Kit, instance of, Video game] [3D Construction Kit, instance of, video game] [3D Construction Kit, platform, Atari ST] [3D Construction Kit, platform, ZX Spectrum] [3D Construction Kit, platform, Amstrad CPC] [3D Construction Kit, platform, DOS]

		[3D Construction Kit, platform, Commodore 64]
[Adam Doboszyński, educated at, Gdańsk University of Technology]	✓	[Adam Doboszyński, educated at, University of Warsaw] [Adam Doboszyński, educated at, Gdańsk University of Technology] [Adam Doboszyński, place of birth, Kraków] [Adam Doboszyński, conflict, Polish-Soviet War] [Adam Doboszyński, country of citizenship, Polish People's Republic] [Gdańsk University of Technology, headquarters location, Kraków]
[Aleksej Pokuševski, place of birth, Belgrade]	☑	[Aleksej Pokuševski, place of birth, Serbia] [Aleksej Pokuševski, date of birth, 2001]
[Alexander Chayanov, occupation, economist]	☑	[Alexander Chayanov, occupation, Economics] [Alexander Chayanov, place of death, Alma-Ata] [Alexander Chayanov, country of citizenship, Russian Empire] [Alexander Chayanov, place of birth, Moscow] [Alexander Chayanov, date of death, October 3, 1937] [Alexander Chayanov, date of birth, 17 January 1888]
[Allan Cup, inception, 1909]	✓	[Allan Cup, inception, 1909] [Allan Cup, sport, ice hockey]
[America Goes Over, publication date, 1918]	✓	[America Goes Over, publication date, 1918] [America Goes Over, genre, silent] [America Goes Over, genre, propaganda film] [America Goes Over, genre, documentary] [American Expeditionary Forces, conflict, World War I] [John J. Pershing, conflict, World War I] [America Goes Over, production company, Committee on Public Information]
[Anatole Shub, occupation, journalist]	✓	[Anatole Shub, occupation, Author] [Anatole Shub, occupation, journalist] [Anatole Shub, occupation, research] [Anatole Shub, occupation, editor] [Anatole Shub, occupation, news director] [Anatole Shub, occupation, public opinion analyst] [Anatole Shub, date of death, July 2, 2006] [Anatole Shub, place of birth, New York City] [Anatole Shub, date of birth, 1928-05-19] [Anatole Shub, date of birth, May 19, 1928] [Anatole Shub, educated at, Columbia University Graduate School of Journalism]
[Anchorage Press, inception, 1992]	✓	[Anchorage Press, inception, 1992]
[Anke Kracke, place of birth, Mainz]	✓	[Anke Kracke, place of birth, Mainz, Germany] [Anke Kracke, place of birth, Mainz] [Anke Kracke, employer, Max Planck Institute for Nuclear Physics] [Anke Kracke, date of birth, 8 September 1983]
[Annapoorna Studios, headquarters location, Hyderabad]	✓	[Annapoorna Studios, headquarters location, Hyderabad, India] [Annapurna Studios, inception, 1951] [Annapoorna Studios, headquarters location, Hyderabad]
[Apt Pupil, publication date, 1982]	✓	[Apt Pupil, publication date, 1982] [Stephen King, notable work, Apt Pupil] [Apt Pupil, author, Stephen King]

[archdiocese of Montreal, headquarters location, Montreal]	☑	[archdiocese of Montreal, headquarters location, Montreal, Canada]
[Avadh Akhbar, founded by, Munshi Nawal Kishore]	✓	[Avadh Akhbar, founded by, Munshi Nawal Kishore] [Munshi Nawal Kishore, languages spoken, written or signed, Urdu]
[Bangsamoro Government Center, inception, 1976]	✓	[Bangsamoro Government Center, inception, 1976] [Bangsamoro Government Center, located in the administrative territorial entity, Autonomous Region in Muslim Mindanao]
[Banjo-Pilot, developer, Rare]	✓	[Banjo-Pilot, developer, Rare]
[Bank of Central African States, headquarters location, Yaoundé]	✓	[Bank of Central African States, headquarters location, Yaoundé] [Banque Centrale des États de l'Afrique Équatoriale et du Cameroun, headquarters location, Paris] [Banque Centrale des États de l'Afrique Équatoriale et du Cameroun, instance of, central bank] [Banque Centrale des États de l'Afrique Équatoriale et du Cameroun, inception, 1955]
[Bard High School Early College Newark, inception, 2011]	✓	[Bard High School Early College Newark, inception, 2011] [Bard High School Early College Newark, located in the administrative territorial entity, Newark, New Jersey] [Bard High School Early College Newark, school district, Newark Public Schools]
[Barry Lee Myers, employer, AccuWeather]	✓	[Barry Lee Myers, employer, AccuWeather] [Barry Lee Myers, employer, U.S. National Weather Service] [Barry Lee Myers, employer, Pennsylvania State University] [Barry Lee Myers, educated at, Pennsylvania State University] [Barry Lee Myers, educated at, Boston University School of Law]
[Belfer Center for Science and International Affairs, founded by, Paul M. Doty]	✓	[Belfer Center for Science and International Affairs, founded by, Paul M. Doty] [Belco Oil & Gas Corporation, founded by, Robert A. Belfer] [Belfer Center for Science and International Affairs, inception, 1973] [Belfer Center for Science and International Affairs, named after, Robert A. Belfer]
[Ben Hur Lampman, occupation, poet]	✓	[Ben Hur Lampman, occupation, Journalist] [Ben Hur Lampman, occupation, essayist] [Ben Hur Lampman, occupation, poet] [Ben Hur Lampman, place of death, Portland, Oregon] [Ben Hur Lampman, date of birth, August 12, 1886] [Ben Hur Lampman, country of citizenship, U.S.] [Ben Hur Lampman, date of death, January 24, 1954] [Ben Hur Lampman, place of birth, Barron, Wisconsin]
[Captain Ron, publication date, 1992]	✓	[Captain Ron, publication date, 1992] [Captain Ron, director, Thom Eberhardt]
[Catarina Ligendza, place of birth, Stockholm]	✓	[Catarina Ligendza, place of birth, Stockholm] [Catarina Ligendza, date of birth, 18 October 1937]
[CBLA-FM, owned by, Canadian Broadcasting Corporation]	✓	[CBLA-FM, owned by, Canadian Broadcasting Corporation] [CBLA-FM, licensed to broadcast to, Toronto] [CBLA-FM, country, Canada] [CBLA-FM, located in the administrative territorial entity, Ontario]

[Chartered Society of Designers, inception, 1930]	✓	[Chartered Society of Designers, inception, 1930]
[Christian Ludwig Gerling, educated at, University of Göttingen]	✓	[Christian Ludwig Gerling, educated at, University of Göttingen] [Christian Ludwig Gerling, attended, Johanneum] [Christian Ludwig Gerling, employer, Hamburg University of Göttingen] [Christian Ludwig Gerling, date of death, 15 January 1864] [Christian Ludwig Gerling, date of birth, 10 July 1788] [Christian Ludwig Gerling, employer, University of Marburg] [Christian Ludwig Gerling, doctoral advisor, Carl Friedrich Gauss]
[Chudov Monastery, founded by, Alexius of Moscow]	✓	[Chudov Monastery, founded by, Alexius of Moscow] [Chudov Monastery, dedicated to, Wonder of Archangel Michael in Chonae] [Alexius, title, Metropolitan of Moscow]
[Club of Rome, founded by, Aurelio Peccei]	☑	[Club of Rome, co-founded by, Aurelio Peccei]
[Coral Pink Sand Dunes State Park, inception, 1963]	✓	[Coral Pink Sand Dunes State Park, inception, 1963] [Coral Pink Sand Dunes State Park, located in the administrative territorial entity, Utah]
[Daily Racing Form, headquarters location, New York City]	✓	[Daily Racing Form, headquarters location, New York City] [Daily Racing Form, language, English] [Daily Racing Form, website, www] [Daily Racing Form, founder, Frank Brunell] [Daily Racing Form, founded, 1894] [Daily Racing Form, owner, Affinity Interactive] [Daily Racing Form, inception, 1894]
[DePaul University, headquarters location, Chicago]	☑	[DePaul University, headquarters location, Chicago, Illinois, US] [DePaul University College of Law, located in the administrative territorial entity, Chicago, Illinois] [The Theatre School at DePaul University, located in the administrative territorial entity, Chicago, Illinois]
[DICE, owned by, Electronic Arts]	✓	[DICE, owned by, Electronic Arts] [Electronic Arts, subsidiary, DICE (company)]
[Domenico Cimarosa, occupation, composer]	✓	[Domenico Cimarosa, occupation, composer]
[Electronic Arts, founded by, Trip Hawkins]	✓	[Electronic Arts, founded by, Trip Hawkins] [Trip Hawkins, employer, Apple] [Electronic Arts, headquarters location, Redwood City, California] [Electronic Arts, inception, 1982]
[Enrique Urquijo, place of birth, Madrid]	☑	[Enrique Urquijo, place of birth, Madrid, Spain] [Los Secretos, location of formation, Madrid]
[Ernest G. Roy, occupation, film producer]	☑	[Ernest G. Roy, occupation, Producer] [Ernest G. Roy, date of birth, 1892]
[Explodemon, developer, Curve Studios]	✓	[Explodemon, developer, Curve Studios] [Explodemon!, platform, PlayStation 3] [Explodemon!, genre, Superhero video game] [Explodemon!, genre, Side-scrolling video game] [Explodemon!, platform, Microsoft Windows]

[Flight to Forever, publication date, November 1950]	☑	[Flight to Forever, publication date, 1950] [Wikipedia - Flight to Forever 1950, author, Poul Anderson] [Wikipedia - Flight to Forever 1950, genre, novella]
[Foreign Policy Initiative, headquarters location, Washington]	☑	[Foreign Policy Initiative, headquarters location, US] [Foreign Policy Initiative, instance of, think tank]
[Francisco Javier Mendieta, educated at, National Autonomous University of Mexico] -- duplicates with acronyms	☑	[Francisco Javier Mendieta, graduated from, UNAM] [Francisco Javier Mendieta, graduated from, Télécom ParisTech] [Francisco Javier Mendieta, place of birth, San Luis Potosí] [Francisco Javier Mendieta, place of, San Luis Potosí] [Francisco Javier Mendieta, educated at, National Autonomous University of Mexico] [Francisco Javier Mendieta, date of birth, 1955] [Francisco Javier Mendieta, date of birth, 28 November 1955]
[Francisco Javier Mendieta, place of birth, San Luis Potosí]	☑	[Francisco Javier Mendieta, place of birth, San Luis Potosí, Mexico] [Francisco Javier Mendieta, date of birth, 28 November 1955] [Francisco Javier Mendieta, date of birth, 1955] [Francisco Javier Mendieta, educated at, National Autonomous University of Mexico] [Francisco Javier Mendieta, place of, San Luis Potosí]
[Francisco Vallejo Pons, place of birth, Es Castell]	☑	[Francisco Vallejo Pons, place of birth, Es Castell, Menorca, Spain] [Francisco Vallejo Pons, date of birth, 21 August 1982] [Francisco Vallejo Pons, title of chess person, chess grandmaster]
[Franz Joseph Gläser, place of birth, Obergeorghenthal]	✓	[Franz Joseph Gläser, place of birth, Obergeorghenthal, Bohemia, Austrian Empire] [Franz Joseph Gläser, place of birth, Obergeorghenthal] [Franz Joseph Gläser, date of birth, 19 April 1798] [Obergeorghenthal, historical region, Bohemia]
[Frederick T. Melges, occupation, psychiatrist]	☑	[Frederick T. Melges, occupation, Psychiatry] [Frederick T. Melges, date of death, 29 July 1988] [Frederick T. Melges, date of birth, 2 December 1935]
[Georgie Ripper, place of birth, London]	✓	[Georgie Ripper, place of birth, London] [Georgie Ripper, date of birth, 1977]
[Get Outta My Way, publication date, 27 September 2010]	✓	[Get Outta My Way, publication date, 27 September 2010] [Get Outta My Way, lyrics by, Damon Sharpe] [Get Outta My Way, lyrics by, Lucas Secon] [Get Outta My Way, record label, Parlophone] [Aphrodite, instance of, studio album] [Aphrodite, publication date, 2010] [Get Outta My Way, performer, Kylie Minogue] [Get Outta My Way, lyrics by, Mich Hansen] [Get Outta My Way, lyrics by, Daniel Davidsen] [Get Outta My Way, lyrics by, Peter Wallevik] [Get Outta My Way, lyrics by, Stuart Price]
[GNOME Web, developer, the GNOME project]	✓	[GNOME Web, developer, The GNOME Project] [GNOME Web, developer, Marco Pesenti Gritti] [GNOME Core Applications, part of, GNOME] [GNOME Web, part of, GNOME] [GNOME Web, part of, GNOME Core Applications]

[GNU Project, founded by, Richard Stallman]	✓	[GNU Project, founded by, Richard Stallman] [Richard Stallman, notable work, GNU Manifesto] [GNU Manifesto, author, Richard Stallman]
[GodFinger, developer, Wonderland Software]	✓	[GodFinger, developer, Wonderland Software] [GodFinger, publisher, ngmoco]
[Gregory Scott Aldering, educated at, Massachusetts Institute of Technology]	✓	[Gregory Scott Aldering, educated at, Massachusetts Institute of Technology] [Gregory Scott Aldering, occupation, astronomer] [Gregory Scott Aldering, date of birth, 1960] [Gregory Scott Aldering, employer, University of California, Lawrence Berkeley Laboratory]
[Grolsch, headquarters location, Enschede]	☑	[Grolsch, headquarters location, Netherlands] [De Grolsch Veste, occupant, Grolsch Brewery] [Grolsch Brewery, headquarters location, De Grolsch Veste]
[Grow Home, developer, Ubisoft Reflections]	✓	[Grow Home, developer, Ubisoft Reflections] [Ubisoft, subsidiary, Ubisoft Reflections] [Grow Home - Wikipedia 2015, platform, Microsoft Windows] [Ubisoft Reflections, owned by, Ubisoft] [Grow Home - Wikipedia 2015, platform, PlayStation 4] [Grow Home - Wikipedia 2015, publication date, September 1,2015] [Grow Home - Wikipedia 2015, publication date, February 4,2015]
[Gustav Ehrhart, occupation, chemist]	✓	[Gustav Ehrhart, occupation, chemist] [Gustav Ehrhart, specialized in, organic chemistry] [Gustav Ehrhart, worked at, Hoechst AG] [Gustav Ehrhart, known for, the ☑thesis of methadone]
[Hans Heinrich Euler, educated at, University of Leipzig]	✓	[Hans Heinrich Euler, date of birth, 6 October 1909] [Hans Heinrich Euler, educated at, University of Leipzig] [University of Leipzig, has notable alumni, Hans Heinrich Euler] [Hans Heinrich Euler, place of birth, Merano] [Hans Heinrich Euler, date of death, 23 June 1941] [Hans Heinrich Euler, date of birth, 6 October 1909]
[Hans Hinkel, educated at, University of Bonn]	✓	[Hans Hinkel, educated at, University of Bonn] [Hans Hinkel, educated at, University of Bonn] [Hans Hinkel, member of political party, Nazi Party] [Hans Hinkel, place of birth, Worms] [Hans Hinkel, country of citizenship, Germany] [Hans Hinkel, date of birth, 1901]
[Istvan Bors, publication date, 1939]	☑	[Istvan Bors 1939, publication date, 1939] [Istvan Bors, publication year, 1939] [Istvan Bors, type, movie]
[István Rév, employer, Central European University]	✓	[István Rév, employer, Central European University] [István Rév, employer, Vera and Donald Blinken Open Society Archives]
[Jean-Michel Coron, educated at, University Pierre et Marie Curie]	☑	[Jean-Michel Coron, educated at, Pierre and Marie Curie University] [Jean-Michel Coron, educated at, École Polytechnique] [Jean-Michel Coron, doctoral advisor, Haïm Brezis] [Jean-Michel Coron, occupation, mathematician] [Jean-Michel Coron, employer, University Pierre et Marie Curie]

[John Abell, occupation, composer]	✓	[John Abell, occupation, countertenor] [John Abell, occupation, composer] [John Abell, occupation, lutenist] [Lady Frances Knollys, married to, John Abell] [John Abell, place of birth, Aberdeenshire] [John Abell, date of birth, 1653]
[John Simpkins, educated at, Harvard University]	✓	[John Simpkins, educated at, public schools of Yarmouth] [John Simpkins, educated at, St. Mark's School] [John Simpkins, educated at, Harvard University] [John Simpkins, graduated from, Harvard University]
[Journal of Phycology, inception, 1965]	✓	[Journal of Phycology, inception, 1965] [Journal of Phycology, instance of, scientific journal] [Journal of Phycology, main subject, phycology] [Journal of Phycology, publisher, John Wiley & Sons, Inc.]
[Jupp Kapellmann, place of birth, Bardenberg]	✓	[Jupp Kapellmann, place of birth, Bardenberg] [Jupp Kapellmann, date of birth, 19 December 1949] [Jupp Kapellmann, member of sports team, 1. FC Köln] [Jupp Kapellmann, member of sports team, TSV 1860 Munich] [Jupp Kapellmann, date of birth, 1949] [Jupp Kapellmann, member of sports team, FC Bayern Munich] [Jupp Kapellmann, member of sports team, Alemannia Aachen] [Jupp Kapellmann, league, Bundesliga]
[Kalashnikov Concern, headquarters location, Izhevsk]	✓	[Kalashnikov Concern, headquarters location, city of Izhevsk] [Kalashnikov Concern, headquarters location, Khamovniki District] [Kalashnikov Concern, headquarters location, Izhevsk]
[Keldon Johnson, educated at, Oak Hill Academy]	✓	[Keldon Johnson, educated at, Oak Hill Academy] [Keldon Johnson, educated at, Kentucky] [Oak Hill Academy, located in the administrative territorial entity, Mouth of Wilson, Virginia]
[Kostas Bakoyannis, occupation, politician]	☑	[Kostas Bakoyannis, occupation, Leader of the Opposition of Athens] [Kostas Bakoyannis, occupation, Mayor of Athens] [Kostas Bakoyannis, occupation, Regional Governor of Central Greece] [Kostas Bakoyannis, occupation, Mayor of Karpenisi]
[Leonard Sarason, educated at, Yale University]	✓	[Leonard Sarason, educated at, Yale University] [Leonard Sarason, educated at, New York University] [Leonard Sarason, studied, music composition] [Leonard Sarason, studied, Mathematics]
[Les Misérables, publication date, 1862]	✓	[Les Misérables, publication date, 1862] [Les Misérables, country of origin, France] [Victor Hugo, native language, French] [Les Misérables, author, Victor Hugo] [Victor Hugo, notable work, Les Misérables] [Les Misérables, language of work or name, French]
[Liberal Vannin Party, founded by, Peter Karran]	✓	[Liberal Vannin Party, founded by, Peter Karran] [Liberal Vannin Party, instance of, political party] [Liberal Vannin Party, inception, 2006]

[Line 5 Eglinton, owned by, Metrolinx]	✓	[Line 5 Eglinton, owned by, Metrolinx] [Line 5 Eglinton, located in the administrative territorial entity, Toronto] [Line 5 Eglinton, item operated, Bombardier Flexity Freedom] [Line 5 Eglinton, terminus, Renforth station] [Line 5 Eglinton, instance of, light rail] [Line 5 Eglinton, terminus, Kennedy station] [Line 5 Eglinton, terminus, Laird station] [Line 5 Eglinton, terminus, Mount Dennis station] [Toronto Transit Commission, item operated, Line 5 Eglinton] [Metrolinx, item operated, Line 5 Eglinton] [Toronto Transit Commission, subsidiary, Metrolinx]
[Live Search Academic, developer, Microsoft]	✓	[Live Search Academic, developer, Microsoft] [Live Search Academic, inception, April 11, 2006]
[Loretta Cessor Manggrum, educated at, Cincinnati Conservatory of Music]	✓	[Loretta Cessor Manggrum, educated at, Fisk University] [Loretta Cessor Manggrum, educated at, Capital University] [Loretta Cessor Manggrum, educated at, Chicago Conservatory] [Loretta Cessor Manggrum, educated at, Hughes High School] [Loretta Cessor Manggrum, educated at, University of Cincinnati – College-Conservatory of Music] [Loretta Cessor Manggrum, educated at, Cincinnati Conservatory of Music]
[Maarten Schmidt, employer, California Institute of Technology]	✓	[Maarten Schmidt, employer, California Institute of Technology] [Maarten Schmidt, place of birth, Groningen] [Maarten Schmidt, educated at, University of Groningen] [Maarten Schmidt, educated at, Leiden University] [Maarten Schmidt, date of birth, 28 December 1929] [Groningen, country, Netherlands]
[Mark Surman, employer, Mozilla Foundation]	✓	[Mark Surman, employer, Mozilla Foundation] [Mark Surman, employer, Commons Group] [Mark Surman, employer, telecentre.org]
[Martina Janková, place of birth, Orlová]	✓	[Martina Janková, place of birth, Orlová] [Martina Janková, place of birth, Ostrava] [Martina Janková, voice type, soprano] [Martina Janková, date of birth, 1972]
[Mega Man, developer, Capcom]	✓	[Mega Man, developer, Capcom]
[Monterey Trail High School, inception, 2004]	✓	[Monterey Trail High School, inception, 2004] [Monterey Trail High School, school district, Elk Grove Unified School District] [Monterey Trail High School, located in the administrative territorial entity, Elk Grove, California]
[Mscape, developer, Hewlett Packard]	☑	[Mscape, developer, Hewlett-Packard]
[Mullurkara railway station, owned by, Indian Railways]	✓	[Mullurkara railway station, owned by, Indian Railways] [Mullurkara railway station, located in the administrative territorial entity, Thrissur district]
[Nancy St. Alban, educated at, Syracuse University]	✓	[Nancy St. Alban, educated at, Syracuse University] [Nancy St. Alban, educated at, Baltimore Actors' Theatre Conservatory] [Nancy St. Alban, studied, Musical Theater]

[Nexus Player, developer, Google]	✓	[Nexus Player, developer, Google] [Nexus Player, developer, Asus] [Nexus Player, operating system, Android (operating system)]
[NHL 17, developer, EA Canada]	✓	[NHL 17, developer, EA Canada]
[Ninja Tune, founded by, Matt Black]	✓	[Ninja Tune, founded by, Matt Black] [Ninja Tune, founded by, Jonathan More] [Ninja Tune, headquarters location, London] [Ninja Tune, inception, 1990]
[North Country Cinema, headquarters location, Calgary]	☑	[North Country Cinema, headquarters location, Calgary, Alberta, Canada]
[Novair, headquarters location, Stockholm]	☑	[Novair, headquarters location, Sweden]
[Open Knowledge Foundation, founded by, Rufus Pollock]	✓	[Open Knowledge Foundation, founded by, Rufus Pollock] [Open Knowledge Foundation, inception, 20 May 2004]
[Orbest Orizonia Airlines, headquarters location, Palma de Mallorca]	✓	[Orbest Orizonia Airlines, headquarters location, Edificio Orizonia Corporación] [Orbest Orizonia Airlines, headquarters location, Palma de Mallorca] [Orbest Orizonia Airlines, instance of, airline] [Palma de Mallorca, country, Spain] [Orbest Orizonia Airlines, airline hub, Palma de Mallorca Airport] [Orbest Orizonia Airlines, country, Spain]
[OS/2, developer, IBM]	✓	[OS/2, developer, IBM]
[Paul McMurtry, educated at, Dedham High School]	☑	[Paul McMurtry, graduated from, Dedham High School] [Paul McMurtry, graduated from, Northeastern University] [Paul McMurtry, educated at, Dexter School] [Paul McMurtry, majored in, Management] [Dedham High School, located in the administrative territorial entity, Dedham, Massachusetts] [Dexter School, located in the administrative territorial entity, Dedham, Massachusetts] [Paul McMurtry, place of birth, Dedham, Massachusetts] [Paul McMurtry, date of birth, October 28, 1965] [Paul McMurtry, date of birth, 1965] [Paul McMurtry, member of political party, Democratic Party]
[Perdue Farms, founded by, Arthur Perdue]	✓	[Perdue Farms, founded by, Arthur Perdue] [Perdue Farms, founded by, Pearl Purdue] [Perdue Farms, inception, 1920]
[Piero Betello, place of birth, Rome]	☑	[Piero Betello, place of birth, Rome, Italy] [Piero Betello, member of sports team, Empoli FC] [Piero Betello, member of sports team, Palermo FC] [Piero Betello, member of sports team, AS Roma]
[Pietro Tacchini, educated at, University of Padova]	✓	[Pietro Tacchini, educated at, University of Padova] [Pietro Tacchini, date of death, March 24, 1905] [Pietro Tacchini, place of birth, Modena] [Pietro Tacchini, date of birth, March 21, 1838]
[Princess Crown, developer, Atlus]	✓	[Princess Crown, developer, Atlus] [Princess Crown, designed by, Hiroyuki Tanaka] [Princess Crown, designed by, Hitomi Fukaumi] [Princess Crown, publisher, Atlus]

[Roger Mosey, employer, BBC]	☑	[Roger Mosey, employer, Selwyn College, Cambridge] [Roger Mosey, employer, British Broadcasting Corporation]
[Royal Academy of Dramatic Art, founded by, Herbert Beerbohm Tree]	☑	[Royal Academy of Dramatic Art, founded by, Sir Herbert Beerbohm Tree] [RADA Education in the London Borough of Camden Universities and colleges, inception, 1904]
[Sagano Line, owned by, West Japan Railway Company]	✓	[Sagano Line, owned by, West Japan Railway Company]
[Sappi, headquarters location, Johannesburg]	✓	[Sappi, headquartered in, Johannesburg] [Sappi, headquarters location, Johannesburg]
[Sayyidah Zaynab Mosque, inception, 1990]	✓	[Sayyidah Zaynab Mosque, inception, 1990] [Sayyidah Zaynab Mosque, located in the administrative territorial entity, Sayyidah Zaynab]
[Shannon Walker, employer, NASA]	✓	[Shannon Walker, employer, Rockwell Space Operations Company] [Shannon Walker, employer, NASA] [Shannon Walker, astronaut mission, STS-66]
[Shuttleworth Foundation, founded by, Mark Shuttleworth]	✓	[Shuttleworth Foundation, founded by, Mark Shuttleworth] [Shuttleworth Foundation, inception, January 2001] [Shuttleworth Foundation, inception, 2001]
[Sisters of Providence of Saint Mary-of-the-Woods, founded by, Théodore Guérin]	☑	[Sisters of Providence of Saint Mary-of-the-Woods, founded by, Saint Mother Theodore Guerin] [Sisters of Providence of Saint Mary-of-the-Woods, inception, 1840]
[Sparebanken Vest, inception, 1823]	✓	[Sparebanken Vest, inception, 1823]
[Station to Station, publication date, 23 January 1976]	☑	[Station to Station, publication date, 1976]
[Sunset in Vienna, publication date, 1937]	✓	[Sunset in Vienna, publication date, 1937] [Sunset in Vienna, publication date, 1940] [Sunset in Vienna, director, Norman Walker]
[TEAM2 cluster, headquarters location, Loos-en-Gohelle]	✓	[TEAM2 cluster, headquarters location, Northern France] [TEAM2 cluster, headquarters location, Urban Community of Lille Métropole] [TEAM2 cluster, headquarters location, Loos-en-Gohelle] [Loos-en-Gohelle, located in the administrative territorial entity, Lille Métropole] [TEAM2 cluster, located in the administrative territorial entity, Lille Métropole]
[The Bangkok Recorder, founded by, Dan Beach Bradley]	☑	[The Bangkok Recorder, founder, Dan Beach Bradley] [The Bangkok Recorder, inception, 1844] [The Bangkok Recorder, dissolved, abolished or demolished date, 1867]
[The Bangkok Recorder, inception, 1844]	☑	[The Bangkok Recorder, inception, July 4, 1844]
[The Gay Divorcee, publication date, 1934]	✓	[The Gay Divorcee, publication date, 1934] [The Gay Divorcee, genre, musical film] [The Gay Divorcee, origin country, United States] [The Gay Divorcee, director, Mark Sandrich]
[The Saturn Game, publication date, February 1981]	☑	[The Saturn Game", publication date, February 2, 1981] [The Saturn Game, publisher, Poul Anderson Works] [The Saturn Game, author, Poul Anderson]

[The Sea Fairies, publication date, 1911]	✓	[The Sea Fairies, publication date, 1911] [The Sea Fairies, author, L. Frank Baum] [L. Frank Baum, notable work, The Sea Fairies]
[The Tsuranga Conundrum, publication date, 4 November 2018]	✓	[The Tsuranga Conundrum, publication date, November 4, 2018] [The Tsuranga Conundrum, publication date, 4 November 2018] [The Tsuranga Conundrum, part of the series, Doctor Who]
[Titanfall 2, developer, Respawn Entertainment]	✓	[Titanfall 2, developer, Respawn Entertainment]
[Tokyo Disney Resort, owned by, Oriental Land Company]	☑	[Tokyo Disney Resort, owned by, The Oriental Land Company] [The Walt Disney Company, licensed to, The Oriental Land Company] [Keisei Electric Railway Company, subsidiary, The Oriental Land Company]
[Tony Nocita, place of birth, Winnipeg]	✓	[Tony Nocita, place of birth, Winnipeg, Manitoba, Canada] [Tony Nocita, place of birth, Winnipeg] [Tony Nocita, date of birth, 9 November 1963] [Tony Nocita, date of birth, November 9, 1963] [Winnipeg Fury, headquarters location, Winnipeg]
[TowerFall, developer, Maddy Thorson]	☑	[TowerFall, developer, Maddy Makes Games]
[University of Nevada, headquarters location, Reno]	☑	[University of Nevada, Reno, headquarters location, Reno, Nevada, U.S.] [University of Nevada, Las Vegas, headquarters location, Paradise, Nevada, US] [University of Nevada, Reno Public university, country, U.S.] [University of Nevada, Las Vegas, country, U.S.] [University of Nevada, Reno School of Medicine Medical school, country, U.S.] [University of Nevada, Reno University of Nevada Press, country, U.S.] [University of Nevada, Reno Public university, located in the administrative territorial entity, Reno, Nevada] [Reno, Nevada, country, U.S.] [Paradise, Nevada, country, U.S.]
[Ustaša, founded by, Ante Pavelić]	✓	[Ustaša, founded by, Ante Pavelić] [Ustaša, founded by, Vjekoslav Servatzy] [Ustaša, founded by, Slavko Kvaternik]
[Vincent Baggetta, place of birth, Paterson]	☑	[Vincent Baggetta, place of birth, Paterson, New Jersey] [Vincent Baggetta, occupation, actor] [Vincent Baggetta, date of birth, 1937] [Vincent Baggetta, date of birth, December 7, 1937] [Paterson, New Jersey, country, U.S.] [Vincent Baggetta, country of citizenship, U.S.]
[Vittoria Puccini, occupation, actress]	✓	[Vittoria Puccini, occupation, Actress] [Vittoria Puccini, date of birth, 18 November 1981] [Florence, country, Italy] [Vittoria Puccini, place of birth, Florence] [Vittoria Puccini, country of citizenship, Italy]
[WARM-FM, owned by, Cumulus Media]	☑	[WARM-FM, owned by, Cumulus Media Inc.] [WARM-FM, licensed to broadcast to, York, Pennsylvania] [WARM-FM, radio format, adult contemporary music]

[WCEM-FM, owned by, Draper Holdings Business Trust]	✓	[WCEM-FM, owned by, Draper Holdings Business Trust] [WCEM-FM, licensed to broadcast to, Cambridge, Maryland]
[WebKit, developer, Apple]	☑	[WebKit, developer, Apple Inc]
[Welsh Courts Act 1942, publication date, 1942]	✓	[Welsh Courts Act 1942, publication date, 1942] [Welsh Courts Act 1942, publication date, October 22nd, 1942]
[William Parmenter, place of birth, Boston]	✓	[William Parmenter, place of birth, Boston] [William Parmenter, date of death, February 25, 1866] [William Parmenter, date of birth, March 30, 1789]
[WMGK, owned by, Beasley Broadcast Group]	✓	[WMGK, owned by, Beasley Broadcast Group] [WMGK, licensed to broadcast to, Philadelphia, Pennsylvania]
[Wolfgang M. Schmidt, employer, University of Colorado at Boulder]	✓	[Wolfgang M. Schmidt, employer, University of Colorado at Boulder] [Wolfgang M. Schmidt, field of study, number theory] [Wolfgang M. Schmidt, received PhD from, University of Vienna] [Wolfgang M. Schmidt, PhD supervisor, Edmund Hlawka] [Wolfgang M. Schmidt, member of, Austrian Academy of Sciences] [Wolfgang M. Schmidt, member of, Polish Academy of Sciences] [Wolfgang M. Schmidt, place of birth, Vienna] [Wolfgang M. Schmidt, occupation, mathematician] [Wolfgang M. Schmidt, date of birth, 3 October 1933] [Wolfgang M. Schmidt, educated at, University of Vienna]
[WordPress, developer, Automattic]	✓	[WordPress, developer, Automattic]
[WordPress.com, owned by, Automattic]	✓	[WordPress.com, owned by, Automattic] [WordPress.com, owned by, Automattic, Inc]
[Y Bham Enuol, place of birth, Buôn Ma Thuột]	✓	[Y Bham Enuol, place of birth, Buôn Ma Thuột] [Y Bham Enuol, place of birth, Đắk Lắk Province]

C Source Code

C.1 Directory Structure

The directory structure and the instructions for the development environment setup are available at <https://github.com/ntcho/LociGraph>.

C.2 Server: Agent Pipeline

The source code for the agent pipeline is available at <https://github.com/ntcho/LociGraph/tree/main/src/server>.

C.3 Client: Browser Extension

The source code for the browser extension is available at <https://github.com/ntcho/LociGraph/tree/main/src/client>.