Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Causal Graph Identification by Large Language Models

Studente/i	Relatore
Piqué Gregorio	**Antonucci Alessandro**
	Correlatore
	Zaffalon Marco
	Committente
Corso di laurea	Codice progetto
Ingegneria informatica	**C10681**
Anno	
2023	

Data

26 agosto 2023

Indice

Abstract

Advances in causal inference is vital across multiple fields and contexts. A correct and complete understanding of the causal relationships behind the system of interest is a fundamental requirement for making accurate decisions. Several methods and techniques can be used to identify causal relationships with the task called causal discovery, but many of these approaches present different flaws and weaknesses. *Large Language Models* (LLMs) can be used as a new assistant to aid human efforts and contributing to the task of causal analysis. This project aims to evaluate the ability of LLMs in identifying causal relationships and causal graphs from natural language texts. We will also present a set of techniques that can be used to improve the accuracy of LLM results. The project required the implementation of a software infrastructure to collect textual data and interact with the GPT API to process it to conduct causal discovery. The results showed that LLMs provide non-trivial contribution in helping with the identification of causal graphs. The application of LLMs to tasks of this nature is still in its early stages and has some limitations, but it has achieved some promising results and revealed new opportunities.

Riassunto

I progressi nell'inferenza causale sono fondamentali in diversi campi e contesti. Una comprensione corretta e completa delle relazioni causali alla base del sistema di interesse è un requisito fondamentale per prendere decisioni accurate. Per identificare le relazioni causali si può usare l'operazione chiamata "scoperta causale", applicando diversi metodi e tecniche, ma molti di questi approcci presentano vari difetti e debolezze. I modelli linguistici di grandi dimensioni o *Large Language Models* (LLM) possono essere utilizzati come un nuovo assistente per aiutare gli sforzi umani e contribuire al compito dell'analisi causale. Questo progetto mira a valutare la capacità degli LLM di identificare relazioni e grafi causali da testi in linguaggio naturale. Verrà inoltre presentata una serie di tecniche che possono essere utilizzate per migliorare l'accuratezza dei risultati degli LLM. Il progetto ha richiesto l'implementazione di un'infrastruttura software per raccogliere dati testuali e interagire con l'API GPT per elaborarli e condurre l'operazione di scoperta causale. I risultati hanno dimostrato che gli LLM forniscono un contributo non banale nell'identificazione di grafi causali. L'applicazione degli LLM a compiti di questa natura è ancora agli inizi e presenta varie limitazioni, ma ha ottenuto risultati promettenti e ha rivelato nuove opportunità.

Progetto assegnato

Descrizione

Il progetto consiste primariamente in uno studio empirico delle possibilità di apprendere grafi causali mediante query su LLMs (large language models). Il lavoro richiede lo sviluppo dell'infrastruttura software (codice Python che interagisce con API) per l'apprendimento del grafo da testi in linguaggio naturale. Una serie di benchmark esistenti verranno usate per la validazione e per il confronto contro altre tecniche allo stato dell'arte.

Compiti

Sviluppo codice. Benchmarking. Valutazione risultati e sintesi.

Obbiettivi

Confronto con lo stato dell'arte.

Tecnologie

Python + API GPT.

Capitolo 1

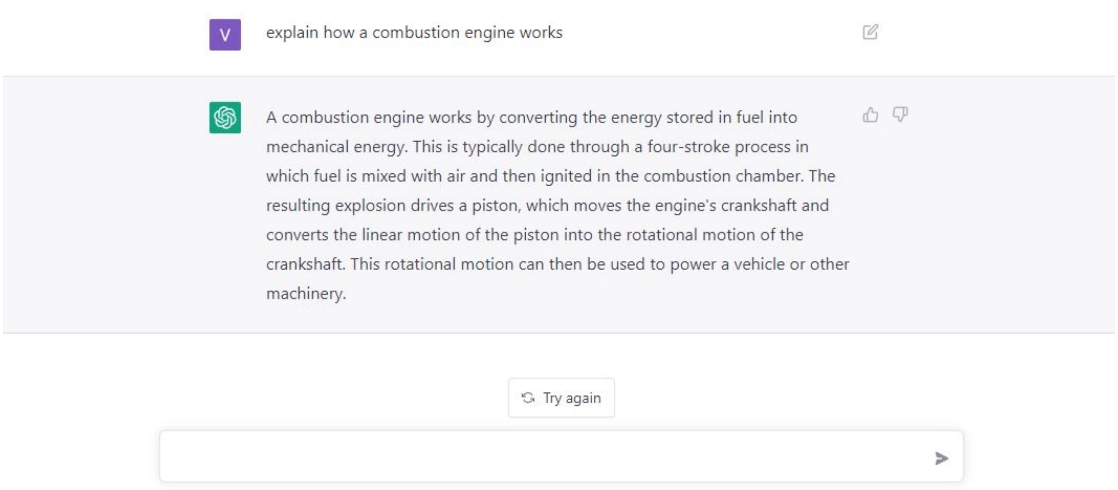
Introduction

Advances in causal inference is crucial for many fields and contexts. In the realm of artificial intelligence and AI systems, a key challenge lies in their predominant reliance on statistical approaches and lack the ability to reason. The need for trustworthy machine learning tools has led to a growing interest in causality as a potential solution to this problem. Causality is the study of cause and effect, and causal understanding is the foundation of sound decision-making [1]. Without it, decisions are likely to be ineffective or even harmful. Causality is a fundamental way of understanding the world, and it can be used to build more accurate and reliable AI systems. Machines equipped with a model of reality, similar to that used in causality, could have the potential to achieve strong AI and artificial general intelligence [2, 3].

Another field where causality is important is the medical one. In medicine, most of the asked research questions are not associational, i.e., modelling statistical relationships between various quantities, but causal in nature; with these questions, researchers try to uncover the cause-and-effect relationships between variables, such as treatments, interventions, and outcomes. These questions cannot be answered from observed data alone and could require specific and expert domain knowledge [2].

Although expert opinion remains one of if not the best tool for causal analysis (e.g., causal discovery for building causal graphs), it can be very time and resource consuming, as the amount of research data becomes larger and larger reaching dimensions that limit the possibility of parsing through the enormity of available evidence. The human factor also increases the likelihood of introducing potential errors or overlooking critical graphs details.

These difficulties could be partially solved by using large language models, which have been trained on vast volumes of textual data [2]. One remarkable example of such a language model is the *Generative Pre-trained Transformer* (GPT) language model. GPT LLMs are designed to understand and generate human-like text based on the given prompts and are accessible, for example, with the GPT API (see Chapter 3.3 for more details) or with the LLM-based chatbot ChatGPT (Figure 1.1).


explain how a combustion engine works

A combustion engine works by converting the energy stored in fuel into mechanical energy. This is typically done through a four-stroke process in which fuel is mixed with air and then ignited in the combustion chamber. The resulting explosion drives a piston, which moves the engine's crankshaft and converts the linear motion of the piston into the rotational motion of the crankshaft. This rotational motion can then be used to power a vehicle or other machinery.

Figura 1.1: ChatGPT prompt example.

1.1 Project Objective

The primary goal of this project is to conduct an empirical study to assess the possibility of performing causal analysis using LLMs.

The project focuses on the operation of causal discovery, which is the task of learning the structure of causal relationships between variables and entities; its output is a directed graph that represents the underlying data-generation process and provides insight into the true causal relationships between variables. The generated graph forms the basis for many, if not all, fundamental tasks in causal analysis, such as effect inference, prediction, and causal attribution [4].

Figure 3.1 shows an example of a simple causal graph that represents the relationship between its entities, which are encoded as graph nodes.

1.2 Document Overview

This section provides a succinct overview of the content and objectives of each chapter in this document.

To start, chapter 2 discusses the state of the art concerning the current causal discovery techniques. Chapter 3 then presents the methods, approaches and tools used in the project. Chapter 4 delves deeper into the implementation details of the developed software for the project. The subsequent chapter 5 presents the benchmark tests and the evaluation metrics used to assess the quality of the results. Chapter 6 then shows and analyses the results achieved and discusses them, addressing also and the limitations found. To conclude, chapter 7 presents the final considerations on the project and topic covered, briefly indicating what was addressed, highlighting the limitations of the project, and then leaving indications of possible future work.

Capitolo 2

State of the Art

This chapter presents the current state-of-the-art methods and techniques used for causal analysis and causal graph identification from data, which have limitations and challenges. A different approach is the knowledge-based method, which focuses on the variables' metadata, rather than their data values. This is the same method adopted by LLMs, who are trained on large amounts of textual data.

2.1 Causal Discovery from Data

The main objective of causal discovery is understanding the underlying cause-and-effect relationships in the system of interest. The current state-of-the-art techniques for causal discovery from data can be divided into two main classes: constraint-based methods and score-based methods [5].

Constraint-based methods use the patterns of conditional independence among variables to deduce or determine the underlying causal relationships within a system, examining how the presence or absence of one variable affects the likelihood of another variable.

Score-based methods, on the other hand, use scoring functions to assess the quality of causal structures and then search for the optimal structure that maximises the assigned score.

Both these methods include different algorithms and techniques. While these have been shown to be effective across multiple applications and scenarios, they are not without challenges. Noise and unobserved confounders can complicate causal relationships identification. Noise is random variation in the data that can obscure the true relationships between variables. Unobserved confounders are variables that are not measured in the study, but that could be affecting the relationship between the variables of interest. Additionally, the limited availability of data further challenges the task of causal discovery.

Moreover, when dealing only with real-world observational data, it is generally not possible to perform causal discovery and find the exact causal graph. The reason relies on the Markov

equivalence class property, where multiple graphs structures are equally likely to be found, given the same data distribution [4, 6].
Different approaches and efforts have tried to overcome these limitations, yet the identification of causal graphs continues to pose challenges, especially when working with real-world observational data, revealing a concerning assessment of their effectiveness [4].
An alternative approach, known as knowledge-based method, can overcome these limitations by adopting a different focus.

2.2 Knowledge-Based Method

The knowledge-based method relies on the metadata associated with variables, rather than their data values. This metadata-based reasoning is typically done by human domain experts when constructing causal graphs, who use their general or specialized domain knowledge and common sense.
However, relying solely on an expert-opinion-based method can prove to be both time and resource-intensive. This approach is also susceptible to errors, as even experts can inadvertently overlook important graph details or make mistakes [2].
Large language models emerge as promising tools for tackling tasks of this nature.

2.3 LLM Approach

Language models provide a fresh perspective to causal discovery by adopting the same metadata-based method: having been trained on vast volumes of textual data, they "reason" through the metadata of variables and the contextual information expressed in natural language. Unlike score-based causal discovery methods, LLMs use their training knowledge combined with additional input data to identify the causal relationships between variables [4].
Multiple recent research papers and publications had a similar interest as this project, wanting to assess the causal capabilities and reasoning abilities of LLMs [1, 2, 4, 7, 8].
While this project concentrated on analyzing the causal capabilities of LLMs specifically for uncovering causal graphs, the other papers had distinct focuses or used alternative approaches. One presents an algorithm that rejects graphs from the same *Markov Equivalence Class*, while controlling the probability of excluding the true graph [1]. One tested various prompt-engineering techniques (see Chapter 3.3.2 for more details on this subject) by using different causation verbs in the prompt messages, incorporating multiple levels of detail in describing the variables and concepts, and referencing various medical authorities [2]. Another analyzed different causal reasoning capabilities in more detail, distinguishing between necessity, sufficiency, normality, and responsibility [4]. Another argued that LLMs are similar to parrots, implying that they only mimic what was seen during the training phase, raising

questions about the extent to which one can truly understand the real world's functioning by merely observing its "shadows" [7]. Lastly, another paper identifies different kinds of causal questions, noting that LLMs seem to perform well primarily with a specific group of these - namely, identifying causal relationships using domain knowledge [8].

Capitolo 3

Methodology and Techniques

This chapter presents the methods, approaches and tools used in the project, which focuses on the operation of causal discovery. This task will be performed starting from natural language, that is textual data, such as scientific papers and research publications. The data will then be processed to extract the main textual entities; a discovery procedure will then be used to find the causal relationship between these entities. The final operation creates the causal graph using the causal relationships found in the previous step and plots the resulting graph.

3.1 Graph Theory and Causality

Causal analysis is an operation consisting of identifying the causal graphs from a given dataset and context, by uncovering the cause-and-effect relationships and dependencies between the variables and entities of the system of interest: this is done by answering questions such as "*Which variables directly affect each other?*" or "*What is the causal directionality between variables?*".
To introduce the causal analysis operation in a more formal manner, the following definitions [9] present the main concepts and assumptions on causality and graph theory.

Definition 1 (Causality)**.** Causality refers to the relationship between cause and effect, where one event (the cause) brings about another event (the effect). It is a fundamental concept in understanding the mechanisms that drive relationships within a system.

Causality relationships in systems can be represented and studied using graphs.

Definition 2 (Graph)**.** A graph $G = (V, E)$ is a mathematical object used to model relationships between objects, represented by a tuple of two sets: a finite set of vertices V and a finite set of edges $E \subseteq V \times V$. A directed graph (DG) G is a graph where the edge (X, Y) is distinct from the edge (Y, X).

Definition 3 (Path)**.** A path π = (X - · · · - Y) is a sequence of vertices such that each vertex is connected to the next vertex in the sequence by an edge. The path can be either directed or undirected. A directed path π = (X $\rightarrow$ · · · $\rightarrow$ Y) is a path where all the edges are directed.

Definition 4 (Cycle)**.** A cycle is a path that starts and ends at the same vertex.

In causality, one of the most used type of graphs is the *Directed Acyclic Graph*.

Definition 5 (Directed Acyclic Graph)**.** A directed acyclic graph (DAG) is a directed graph G that has no cycles.

Definition 6 (Causal Graph)**.** A causal graph G is a graphical description of a system in terms of cause-and-effect relationships, i.e., the causal mechanism.

Causal graphs, usually in the form of directed acyclic graphs, encode contextual knowledge of variables (both observable and unobservable) and their causal dependency. In a directed causal graph, the nodes represent the context entities and variables (e.g., in a medical context they would be symptoms, illnesses, diseases, treatments, medications, outcomes, etc. . . .) while the edges represent the causal relationship between said entities (e.g., a medical treatment can cause a particular outcome or side effect), indicating the direction of causality.

Definition 7 (Causal edge direction)**.** In a causal DAG, the relationships between pairs of entities are represented using graph edges, which can take the form of directed, bidirected, or non-existing edges.

- The **directed edge** (A $\rightarrow$ B) denotes a direct causal dependency between the two features A and B, where A is a direct cause of B, without excluding the possible presence of a common cause of both A and B.
- The **bi-directed edge** (A $\leftrightarrow$ B or both A $\rightarrow$ B and A $\leftarrow$ B) represents a causal relationship where A and B are causally correlated, and the two variables have an unobserved or latent common cause.
- A **non-existent edge** denotes that no causal relationship exists between the two variables.

Definition 8 (Direct and Indirect Cause)**.** For each directed edge (X, Y) $\in$ E, X is a direct cause of Y and Y is a direct effect of X. Recursively, every cause of X that is not a direct cause of Y, is an indirect cause of Y.

Figure 3.1 shows a simple example of a causal graph. In this example, the variable “smoking” is the cause for both “lung cancer” and “tumors”. Neither of these variables is the cause of the other; instead, they share a common cause—namely, “smoking".

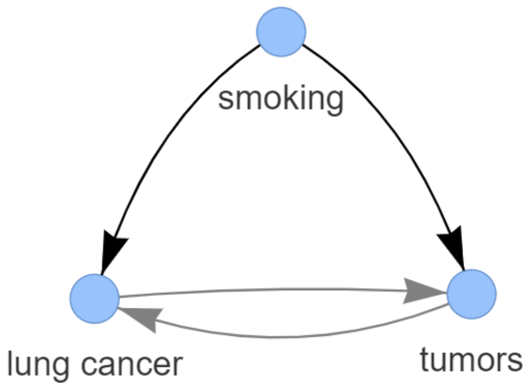


Figura 3.1: Causal graph example.

A crucial property of causal DAGs is their acyclicity: it is vital for ensuring their interpretability and for preserving the causal relationships they represent. This property is fundamental for many reasons: it ensures logical consistency, temporal ordering, identifiability of causal effects, facilitates counterfactual reasoning, and aids in prediction and intervention tasks.

Although causal graphs are mostly represented as acyclic, there are cases where cycles are accepted and even necessary to encode complex dynamics that exist in the system of interest, like for economic scenarios. Figure 3.2 shows a simple example of a case where a cycle is necessary to represent the complex relationship between investments, economic growth, and inflation.

Central items to the causal graphs are the nodes, which represent essential elements within the domain of interest. In the context of this project, the domain pertains to the medical and health-related field, so the nodes would be symptoms, diseases, treatments, outcomes, etc. . . . The data used for constructing these causal graphs is collected in a textual form, from abstracts of medical publications and research papers. For this reason, it is necessary to process the textual data in order to extract the entities. The key elements of the text can be identified and isolated using appropriate *Natual Language Processing* (NLP) operations, such as *Named Entity Recognition* (NER).

3.2 Named Entity Recognition in NLP

NLP is a field of artificial intelligence that makes human language intelligible to machines. It involves the use of algorithms and statistical models to enable computers to understand and process human language. NLP has many applications, including machine translation, sentiment analysis, text summarization, and speech recognition. It is used in a wide range of industries, including healthcare, finance, and marketing [10]. Large language models can be considered among the most advanced and impactful expressions of NLP applications.

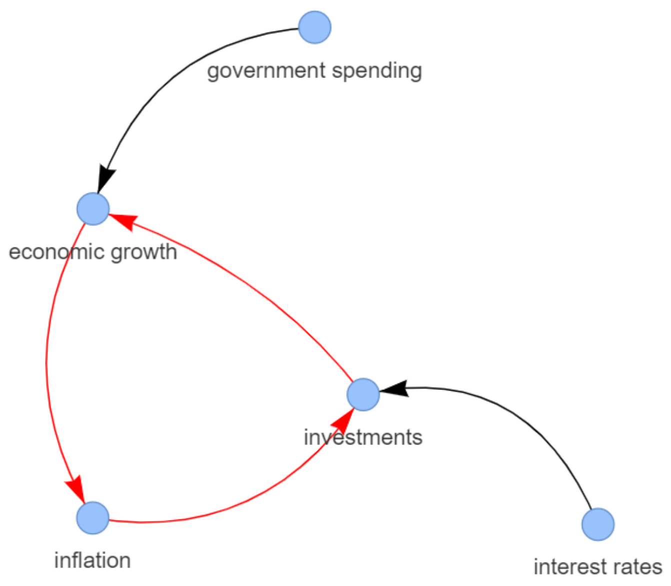


Figura 3.2: Graph example with cycle.

These models are at the forefront of NLP research and technology due to their ability to generate coherent and contextually relevant human-like text.
Named entity recognition (NER) is a crucial NLP task that aims to identify and classify named entities within text. In the context of medical texts, NER plays a vital role in extracting specific medical entities such as diseases, symptoms, treatments, drugs, anatomical terms, and medical procedures. Medical texts pose challenges for NER due to their specialized terminology, which often includes abbreviations and multiple names and entities that are used interchangeably as synonyms. Additionally, the complex language structures found in medical texts, along with the diverse sources from which they originate, further complicate the NER process.
The NER operation can be conducted in several ways, with many tools and packages, such as *spacy* [11], *flair* [12], and the LLM *Bert* [13]. However, to follow the project's general idea of employing LLMs (the GPT model to be specific), this step of NER was performed using the GPT LLM. The model was queried using the available API.

3.3 GPT API

As anticipated, part of the project's activities were performed using the GPT LLM through the GPT API, such as causal relationship identification and the NER operation.
The GPT API is a tool that provides access to OpenAI's GPT models, allowing the integration of natural language processing capabilities into applications. It is a RESTful API and users can send requests by providing a valid API Key. Developers can use any language to

interact with the API, but there are official OpenAI packages and libraries that simplify the development process. For this project, the Python package was used [14].
When using the API, the user specifies the model to use and provides the necessary parameters such as the prompt and optional messages to contextualize the use and behavior of the model, i.e., how the model should answer to requests. The API then processes the request and returns as a response the generated context-aware text based on the provided input.
The GPT API can be employed in a variety of applications and use cases. It can be used to generate conversational agents, draft emails or other pieces of writing, provide language translation, answer questions, or assist with content creation [15].

3.3.1 Using the GPT API

The GPT API is used by specifying the model to use (e.g., GPT-4) and by providing additional messages, including the system message and the user message. These serve as instructions to the model, with the system message being a system level instruction to guide the model's behavior throughout the conversation (e.g., asking the model to answer or to act in a specific way), and the user message functioning as the actual request the model is required to answer.
In particular, the system message is used to contextualize the model and its behavior, to make it more useful and accurate for the required operation: for the project's causal analysis tasks, for example, the system message was "*You are a helpful assistant for causal reasoning and cause-and-effect relationship discovery*", to try steering the output space to more causally consistent answers. This was shown being an effective prompt-engineering technique that results in more accurate answers [4]. Prompt-engineering will be discussed in more detail in the next section 3.3.2.
The system message helps set the behavior of the assistant, by modifying the personality of the assistant or providing specific instructions about how it should behave throughout the conversation. However, this message is optional and the model's behavior without a system message is likely to resemble that of using a generic one, such as "*You are a helpful assistant.*" [15].
Another parameter that can be optionally set is the temperature: this represents the degree of exploration or randomness of the model's output. It ranges from 0 to 2, with a default setting of 1. A higher value (e.g., 0.8) increases creativity and diversity but might be less focused. A lower value (e.g., 0.2) produces more deterministic output following patterns [15, 16]. The default value for the temperature parameter used in this project was 0.2.
Follows a complete example of a GPT API chat completion request that uses the GPT-4 model, and specifies system, assistant, and user messages.

```
openai.ChatCompletion.create(
  model="gpt-4",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the
     World Series in 2020."},
        {"role": "user", "content": "Where was it played?"}
    ],
  temperature=0.3,
)
```

An example of response looks as follows:

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "The 2020 World Series was played in Texas at Globe
    Life Field in Arlington.",
        "role": "assistant"
      }
    }
  ],
  "created": 1677664795,
  "id": "chatcmpl-7QyqpwdfhqwajicIEznoc6Q47XAyW",
  "model": " gpt-4",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 17,
    "prompt_tokens": 57,
    "total_tokens": 74
  }
}
```

In Python, the assistant's reply can be extracted as follows:

```
response['choices'][0]['message']['content']
```

3.3.2 GPT Prompt Engineering

Depending on the task, the GPT LLM can produce satisfactory results when asked to answer a question. However, there are some expedients that have shown to be beneficial and to increase the results accuracy when querying the LLM. These techniques are part of the discipline known as prompt-engineering, which comprises a set of rules and instructions that serve as guidelines to enhance the capabilities of LLMs on a wide range of common and complex tasks [17, 18].

Prompt engineering has emerged as a powerful technique to enhance the performance and control the behavior of language models, particularly large language models, such as the used GPT-3.5 and GPT-4 models. Prompt engineering involves crafting system and user messages that guide the model's responses and shape its output to meet specific requirements [19].

The goal of prompt engineering is to provide contextual cues and instructions to the language model, enabling it to generate more accurate, relevant, and desired responses. By designing prompts, it is possible to tailor the behavior of LLMs, making them more suitable for various tasks, domains, and user needs.

Among the many prompt engineering techniques [16, 17, 19, 20], one of the most important strategies is improving the clarity and precision of the prompt text by providing clear and specific instructions. Delimiters like brackets, tags or quotes can segregate sections within the prompt, aiding in a more organized interpretation of the input. Furthermore, prompting for structured output by specifying the desired response format guides the model in generating well-organized results.

Checking task conditions also ensures the necessary assumptions are met. For instance, the prompt can verify whether essential information is available to complete the task and provide alternative instructions if this information is missing.

The technique called *few-shot prompting* instead involves showing successful task examples to the model before requesting similar ones. This helps the model understand the context better, preparing it to deliver pertinent and accurate responses.

Another principle of prompt engineering consists in providing the model with enough time to "think". This involves requesting the model to answer with a step-by-step explanation of its thought process before providing the final answer. By doing so, the model is directed to work out its own solution rather than rushing to conclusions. This principle applies, for example, when the model needs to verify the accuracy of a given solution: in this case, it is prompted to formulate its solution first and then compare it to the provided one [16, 20].

Appendix B shows the user messages used in the project.

Capitolo 4

Implementation

The following chapter presents the implementation details of the project. The project can be divided in two main steps: data collection and data analysis. The former one consisted in collecting the necessary data for the latter, which can itself be divided into multiple other sub-operations. This section will first present the data collection process, highlighting the usage of the *National Center for Biotechnology Information* (NCBI) API for requesting the necessary textual data. It will then delve into the operations of data processing and causal analysis.

4.1 Scraping

The first step of the project consisted in collecting the necessary textual data for testing the causal discovery capabilities of the GPT LLM.
This involved implementing a series of Python scripts to automate a data acquisition process. Considering the project's focus on medical papers and health-related publications, this general scraping process was adapted to suit the relevant context. As a result, the data was sourced from the PubMed database. PubMed serves as a freely accessible search engine mainly utilizing the *MEDLINE* database, which contains references and abstracts related to life sciences and biomedical subjects [21].
The scraping process only extracts the article abstract and extra details, such as title, publication date, and keywords.

4.1.1 Scraping Pipeline

A pipeline handling the essential operations was created for extracting the necessary textual data from the PubMed database. To automate this extraction process, a Python script was written using the public API provided by the NCBI as interface into its query and database system. The main flow is shown in Figure 4.1. The pipeline allows the user to extract textual data from PubMed by searching for specific terms.

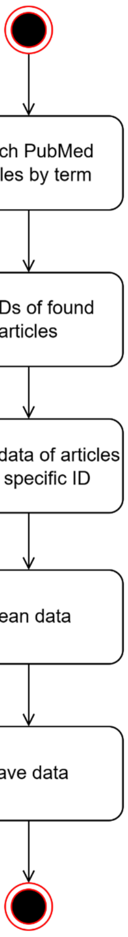

search PubMed articles by term
get IDs of found articles
extract data of articles with specific ID
clean data
save data

Figura 4.1: PubMed data extraction flow.

The pipeline's main operations are handled by the *search_by_terms*, *get_articles_data*, and *clean_data* procedures. Appendix A.1 presents the source code of these functions.

4.1.2 Collecting the Data

The *search_by_terms* procedure is the first operation of the pipeline. As the name suggests, it allows the user to search for articles in the PubMed database containing the specified search terms. These terms are joined as query parameters in the request URL. An API key is also sent in the request URL, to ensure smooth and supported access to the desired resources.

The response is in a xml format, and it is processed to extract all ID numbers of the articles found in the specified NCBI database, which in this case is PubMed.

The implemented function returns the extracted IDs. However, the script allows users to utilize the NCBI *Entrez History* feature, which proves to be significantly more efficient when dealing with tasks that involve searching for or downloading a substantial number of records. This approach helps to streamline the process and optimize the retrieval of records in a more efficient manner, making it possible to upload many IDs or download several hundred records at once [22].

The pipeline then continues with the second step of the data acquisition process, with the *get_articles_data* function. This procedure queries the NCBI for the actual content of the articles with the specified ID.

The NCBI API allows users to query article data with the article ID or by using the Entrez History feature, which can provide a more efficient data retrieval. A URL parameter of the request defines the main data content requested, which, in this case, are the abstracts (*rettype=abstract*).

The returned data is in a xml format, and it is processed and parsed to extract the necessary information. The recovered data from the article includes the abstract, the article ID number, the title, the keywords, and the publication date.

4.1.3 Cleaning the Data

The *clean_data* procedure is the third and last step of the data acquisition pipeline. It performs cleaning operations on the acquired data, such as removing null abstract values, duplicates, and potentially removing data of articles published in a particular date range.

The cleaned data is eventually saved to a file and returned by the scraping pipeline. This data can also serve as input for the causal analysis pipeline, which will process it to extract causal graphs from the obtained abstracts.

4.2 Causal Analysis

After completing the preliminary phase of data collection, the main focus of the project shifted towards the actual analysis operations. The primary objective was to investigate the causal capabilities of LLMs, specifically focusing on causal discovery.

Causal analysis involves the process of revealing the cause-and-effect relationships and dependencies among variables from a provided dataset and context, by answering questions such as "*Which variables directly affect each other?*".

This step of the project involved processing the collected data from the PubMed database to extract information from the abstracts. This consisted of extracting the main named entities within the textual data, such as the names of diseases, drugs, and genes. These named entities were then used to perform the actual causal analysis. The output of the causal relationship discovery process is then used to plot the resulting causal graph.

4.2.1 Causal Analysis Pipeline

The components of the causal analysis process, consisting of various sub-steps, have been integrated into a single operational pipeline, called *causal_discovery_pipeline*, whose general flow is shown in Figure 4.2. These operations include extracting entities from the textual data with the *gpt_ner* function, performing the actual causal analysis on the found entities using the *gpt_causal_discovery* procedure, and ultimately generate the resulting causal graph with the *build_graph* function. Appendix A.2 presents the source code of these functions.

4.2.2 NER: Extracting Medical Entities from Text

As previously mentioned, the second part of the project consisted in working with the collected data. The first step of the operation involved performing Named Entity Recognition on the abstracts, a fundamental procedure to extract and classify named entities. This step was essential for further processing and analysis.

The NER operation was performed using the GPT LLM. To enhance the performance of the Language Model for the task, both the system and user messages were designed accordingly.

The employed system message was "*You are a helpful assistant for Named Entity Recognition of medical texts*" to provide guidance to the model and improve its understanding of the task at hand.

To further aid the model's comprehension, the user message was crafted using the abstract of the medical text, complemented with additional information about the types of entities to be extracted. In this case, since the texts were focused on medical literature and research

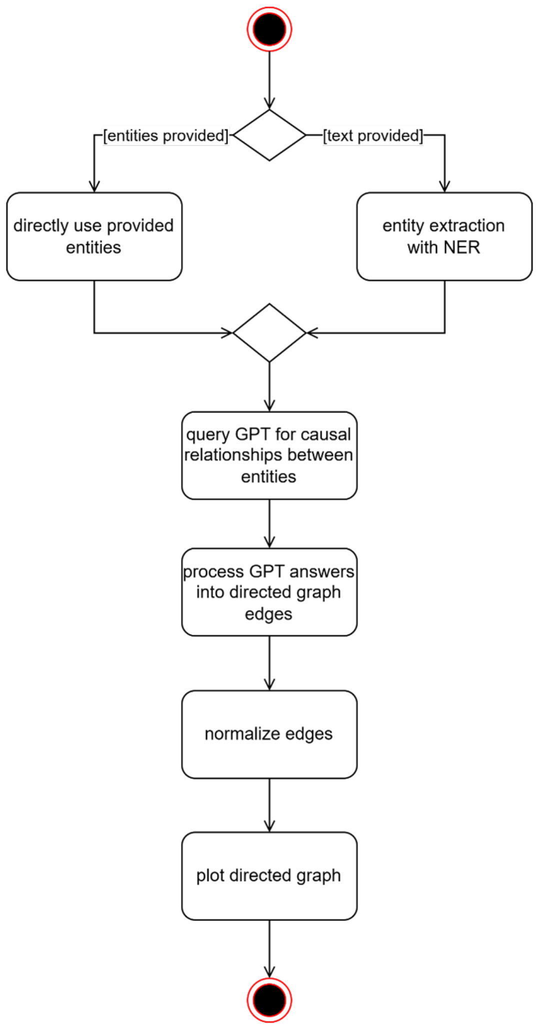

[entities provided]
[text provided]
directly use provided entities
entity extraction with NER
query GPT for causal relationships between entities
process GPT answers into directed graph edges
normalize edges
plot directed graph

Figura 4.2: Causal discovery pipeline flow.

publications, the model was explicitly instructed to identify entities with a particular emphasis on diseases, medications, treatments, symptoms, etc. . . .

The intention of customizing the user message by providing relevant context and specific entity requirements, was to guide the LLM towards producing more accurate and relevant results for the ongoing NER operation. Appendix B.1 shows the full user message.

The result of the *gpt_ner* function is an array containing all the found entities and it is then used for the subsequent causal analysis.

The *causal_discovery_pipeline* also provides the option to include an entity optimization step through the *optimize_entities* procedure: by using the GPT API, the pipeline operation focuses on identifying synonyms, redundant entities, or entities and names that can be used interchangeably. In the generated output, entities with synonymous or similar meanings are matched together. Appendix B.2 shows the full user message used for the entity optimization function.

4.2.3 Causal Discovery

With the completion of the NER operation and the extraction of entities, we now proceed to the central step of the pipeline: the causal discovery operation.

This step consists of the *gpt_causal_discovery* function, which takes the found entities and performs causal discovery. The approach for this operation uses a naïve discovery procedure. This method does not use actual data as input to perform inferences: in this sense, it resembles how humans recall facts by relying on memorized knowledge rather than actively referencing actual data [7]. This approach aims to infer the causal relationship among the various variables by querying the LLM regarding the direction of the pairwise causal relationships for each possible pair combination.

The type of causal relationship between a pair of entities corresponds to the edge orientation in the causal graph: directed edges indicate direct causes, bi-directed edges represent entities that are causally correlated and the two have an unobserved or latent common cause, and non-existent edges indicate the absence of a causal relationship between the variables.

To infer the direction of the causal edge, the pipeline function queries the LLM to determine which cause-and-effect relationship is more likely between the two entities. The system message used for this operation is "*You are a helpful assistant for causal reasoning and cause-and-effect relationship discovery*", to try guiding the output towards more causally consistent answers. On the other hand, the user message introduces the current pair of entities of interest, asking a single question about the direction of the causal dependency. It also requests a step-by-step explanation in response. The possible answers the LLM is expected to choose from are also listed within the user message:

A. "X" causes "Y";

B. "Y" causes "X";

C. "X" and "Y" are not causally related;

D. there is a common factor that is the cause for both "X" and "Y"

To then enhance the accuracy and exploration of cause-and-effect relationships, the prompt uses random verbs of causation when querying the GPT LLM. This approach can be beneficial in terms of coverage of language patterns and potential causal relationships, can reduce the risk of bias that may come from consistently relying on a specific verb, and can encourage the model to explore different relationships between variables, allowing for a more comprehensive analysis of the data [1, 2]. Appendix B.3 shows the full user message used for the causal discovery process.

The LLM is queried for each pair of variables. In a default execution mode, the pipeline examines combinations (without repetition) of all pairs of the identified entities. The total number of queries (one for each pair) is

$$C_k(n) = \binom{k}{n} = \frac{n!}{k!\,(n\,-\,k)!}\,; \tag{4.1}$$

in the case of ten entities, the total queries are $C_2(10) = \binom{2}{10} = \frac{10!}{2!\,(10\,-\,2)!} = \frac{10\,\cdot\,9}{2} = 45.$
The pipeline, however, allows to perform a double test for each pair of variables, checking all potential variations without repetition (i.e., relationship “X” - “Y” and “Y” - “X”). As a result, the LLM is queried twice for each pair of entities, with a total number of variations of

$$V_k(n) = \frac{n!}{(n\,-\,k)!}\,. \tag{4.2}$$

With ten entities, the total queries are $V_2(10) = \frac{10!}{(10\,-\,2)!} = \frac{10!}{8!} = 10\,\cdot\,9 = 90.$
In this case, the pipeline also performs an answer compatibility verification: the response to the query regarding the causal relationship between “X” and “Y” is cross-validated with the response to the query about the relationship between “Y” and “X”, and the two answers must be consistent with each other. This validation process is managed by the *check_invalid_answers* function, which assesses response consistency and distinguishes between valid and invalid edge directions.

The edge direction and causal relationship associated with “invalid” answers are then re-queried using the *correct_invalid_edges* function. In this process, the LLM is queried again by also adding, in the user message, the inconsistent answers obtained earlier, seeking the most likely relationship between the given variables. The newly acquired answers are then appended to the previously identified "valid" edges.

The output of this pipeline step involving the *gpt_causal_discovery* function is an array containing the type of causal relationship between each pair of entities. Appendix B.4 shows the user message used for the invalid edge correction process.

4.3 Plotting the Causal Graph

The upcoming and final operations of the project involve preparing for the construction of the resulting causal graph, using libraries and packages such as *NetworkX*, *Pyvis*, and *graph-tool*. Appendix A.3 presents the source code of the used functions.

4.3.1 Graph Preprocessing

Before plotting the graph, the pipeline performs an intermediate operation of edge and node preprocessing. This step assists the following ones by decoding the LLM answers and converting them into sets of nodes and normalized directed edges, represented in the form of “X $\rightarrow$ Y”.
The main procedure responsible for this operation is the *preprocess_edges* function. This function generates various components: a set consisting of all graph nodes (entities previously extracted), an array containing the normalized directed edges and one containing all bidirectional edges.
The *preprocess_edges* function relies on another procedure for the normalization of edges, which is the *normalize_edge_direction* function. This procedure takes as input the nodes involved and the LLM’s response regarding their causal relationship. The function processes the output of the LLM and returns the corresponding edge in the form of “X $\rightarrow$ Y”, representing the causal dependency between the nodes.
An additional step is taken before plotting the graph, where a check is performed to determine whether the resulting graph is acyclic.

4.3.2 Cycle Check

The resulting graph should be a DAG, which is a directed graph without cycles. In the context of causal analysis and causality in general, the acyclic property of graphs is crucial to maintain the logical meaning and coherence encoded within the graph. The absence of cycles ensures that there are no circular dependencies or contradictory relationships, allowing for a clear and meaningful representation of causality in the system.
The *find_cycles* function is the dedicated procedure designed to determine whether the constructed graph contains any cycles. It accepts an array of nodes and an array of edges as input parameters, which together represent the graph. The function makes use of the graph-tool Python package, a highly efficient module for graph manipulation and analysis. The underlying components of this package are primarily implemented in C++ to optimize performance [23].
The cycles are identified using the *all_circuits* function from the graph-tool package. This function is based on the Johnson’s algorithm for finding all elementary circuits of a directed graph; it is similar to other algorithms (e.g., by Tarjan [24]), but it is faster because it consi-

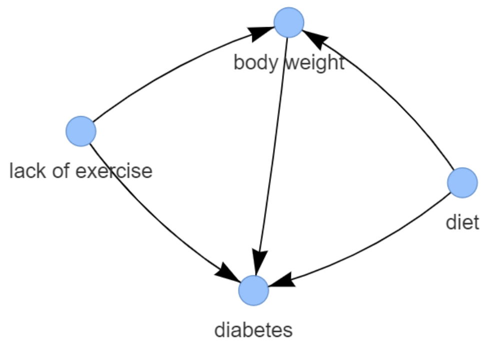


Figura 4.3: Interactive graph generated with Pyvis for the conducted causal analysis.

ders each edge at most twice between any one circuit and the next in the output sequence. The algorithm is time bounded by $O((n + e)(c + 1))$ and space bounded by $O(n + e)$, for a graph with *n* nodes, *e* edges and *c* cycles [25].

In a directed graph, the number of potential cycles has a tendency to grow exponentially as the number of nodes increases. As a result, even relatively small graphs can contain a substantial number of them. Since listing and returning all cycles of a graph is not the main focus of this project, the *find_cycles* function only returns a symbolic number of the first 100 found, assuming there are any.

In case cycles exist, they are represented as lists of graph nodes and returned as an output parameter of the *find_cycles* function.

4.3.3 Plotting the Graph

The operation for plotting the causal graph is the last step of the casual discovery pipeline, and it is processed by the *build_graph* procedure. This function is designed to construct and visualise a directed graph using libraries for graph and network creation, manipulation, and analysis, such as the NetworkX and Pyvis libraries. NetworkX is dedicated to general-purpose graph operations [26], while Pyvis serves as a visualisation library suitable for generating interactive network graphs [27].

Within a graph, bidirected edges are coloured in grey, and the function allows for highlighting cycles, as well as creating interactive plots. Cycles within the graph can be highlighted by coloring relevant edges in red. It also supports both static and interactive modes of graph presentation, simplifying the visualisation and analysis of entity relationships. The final interactive graph is then exported as an *.html* file.

Figure 4.3 presents an example of the outcome of the conducted causal analysis. It displays the resulting interactive causal graph, which has been plotted using the Pyvis package.

4.4 Running the Process

A separate Python script called *causal_analysis.py* was implemented to allow running the entire process from a terminal. The script can be used to execute both the scraping and causal discovery operations, either sequentially (using the data extracted from the scraping step as input to the causal discovery step) or independently.
The script also allows running the causal discovery on data specified by the user or on the benchmarks, where the user can also choose to use the random baseline algorithm or the GPT LLM.
The following examples show two of the possible commands to run the causal analysis process on a specified data file (defined with the *data-path* option) or on the benchmark tests using the GPT LLM (instead of the baseline algorithm).

```
$ python causal_llms.py c --data-path=../data/abstracts.csv
$ python causal_llms.py b --algorithm=gpt
```

Appendix D shows all command line actions and options.

Capitolo 5

Benchmarks and Metrics

This chapter introduces the benchmarks used to evaluate the capabilities of LLMs. These include tests for two types of tasks: pairwise causal relationships and full graph discovery. Additionally, real-world text evaluations are conducted, focusing specifically on abstracts from medical papers, which are relevant to this project. The chapter then shows the set of evaluation metrics used to assess the quality of the achieved results, like commonly used *precision*, *recall*, *F1 score*, and *Structural Hamming Distance* (SHD).

Because of its relatively recent popularity, this subject has witnessed many contributions over the past few months. Many publications and research papers have tested the general capabilities of LLMs using standardized exams and tests written to assess human aptitude and knowledge across various domains. For causal reasoning capabilities, researchers have used widely known and established benchmarks with datasets from multiple domains, including medicine and climate science [4].

5.1 Pairwise Causal Relationship Discovery

In the context of causal discovery abilities, these benchmark datasets mainly consist of lists of variable pairs, where each pair represents a causal relationship that can be encoded as a directed edge in a causal DAG.

The assessment of LLMs' causal discovery capabilities involves tasks that focus on identifying pairwise causal relationships, determining whether variable *A* causes variable *B* or vice-versa. These tasks involve both well-known scenarios that an average non-expert can correctly address using common sense and basic field knowledge (e.g., Tubingen cause-effect pairs dataset [28]), as well as more specialized domains that require expertise in a specific field to ensure accurate understanding and interpretation (e.g., Neuropathic pain dataset [29]). Table 5.1 [4] shows some examples of the cause-effect pairs and respective domains of interest from the the Tubingen benchmark, where the task is to determine which variable is the cause and which one the effect.

Variable A	Variable B	Domain
Age of Abalone	Shell weight	Zoology
Cement	Compressive strength of concrete	Engineering
Alcohol	Mean corpuscular volume	Biology
Organic carbon in soil	Clay content in soil	Pedology
Photosynthetic Photon Flux Density	Net Ecosystem productivity	Physics
Drinking water access	Infant mortality	Epidemiology

Tabella 5.1: Example cause-effect pairs from the Tubingen benchmark.

As previously mentioned, it has been noted that prompt engineering significantly increases the accuracy of results when querying the LLM for causal dependencies and edge directions [18]. Furthermore, using advanced Language Models, such as GPT-4, along with these prompt engineering techniques results even in higher accuracy.

5.2 Full Causal Graph Identification

Since the primary objective of this project is to evaluate the abilities of LLMs in identifying complete causal graphs, the LLM was tested using slightly different benchmarks compared to the ones mentioned earlier.

Extending the task from simple identification of pairwise causal relationships to full graph discovery introduces additional challenges that are not present in the former task. These include, for example, the need to avoid introducing edges between unrelated variables and distinguishing between direct and indirect causes [4].

The adopted strategy, as discussed in the previous chapters, involves enumerating all possible pairs of variables and performing the pairwise test for each pair combination.

For this project, the LLM was tested against existing causal graphs, which served as benchmarks representing the ground truth. The graphs used as ground truth [2, 30] predominantly revolve around medical and health-related subjects, as the project focused on identifying causal relationships and uncovering causal graph structures within the medical context.

Figure 5.1 shows one of ground truth causal graphs used in the tests. Appendix C.1 shows all other ones.

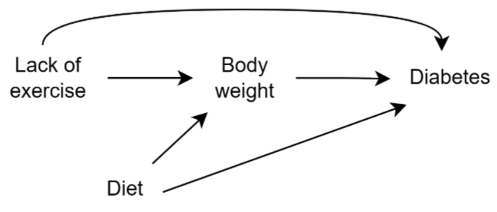


Figura 5.1: 'Diabetes' benchmark.

Metric	**Description**
Missing edges	Number of edges that are present in the ground truth graph but not in the generated one
Extra edges	Number of edges that are present in the generated graph but not in the ground truth one
Correctly directed edges	Number of edges present in the generated graph that were correctly directed
Incorrectly directed edges	Number of edges present in the generated graph that were incorrectly directed
Structural hamming distance	Sum of missing edges, extra edges, and incorrectly directed edges
Precision	Measures how many of the identified causal relationships are correct out of the total relationships identified. $Pr = \frac{TP}{TP+FP}$
Recall	Measures the ability to identify all actual causal relationships. $Re = \frac{TP}{TP+FN}$
F1 score	Harmonic mean of precision and recall. $F1 = 2 \cdot \frac{Pr \cdot Re}{Pr+Re}$
Precision-Recall Curve	Depicts the trade-off between the precision and recall of the identified causal relationships.
Area Under PR Curve	Quantifies the overall performance by summarizing the precision-recall trade-off across different thresholds.

Tabella 5.2: Evaluation metrics.

5.3 Evaluation Metrics

Various evaluation metrics are used to assess the quality of the obtained causal discovery results. These metrics aim to identify shared patterns between the ground truth model and the one generated from the process. Given that the ground truth when dealing with causality and causal discovery is commonly represented in a graph form (e.g., DAGs), these metrics are also related with network ones.
These include commonly used metrics like precision, recall, F1 score, accuracy, Structural Hamming Distance (SHD) (calculated with the *causal-discovery-toolbox* package [31]) and more. Table 5.2 [32] lists the evaluation metrics used for the benchmark tests.

Capitolo 6

Results and Discussion

This chapter presents the results of the causal discovery tests conducted on benchmarks and real medical texts, and addresses the limitations found. The results are discussed, showing how well the tests performed compared to the ground truth graphs. Among the metrics used to evaluate the performance of the tests are precision, recall, and F1 score.

6.1 Causal Discovery from Benchmark Tests

The benchmark evaluation allows to quantify the capabilities of the LLM by comparing its predictions against a ground-truth reference, with commonly used evaluation metrics, like accuracy, precision, and recall.

The benchmarks are run with different LLM-based methods and algorithms. An algorithm returning either edge at random for each pair of entities was used as a baseline. Table 6.1 shows the results of the different methods and algorithms applied to the benchmark tests. Appendix C.2 shows additional results, with the previously presented evaluation metrics.

As expected, the baseline algorithm (indicated by the "Random" row in the table) presents the poorest performance, with an average structural hamming distance exceeding 8 errors, a precision of 0.33, a recall of 0.38 and a resulting F1 score of approximately 0.35.

The *gpt-3.5-turbo* model instead shows better results, with an average SHD that is half of the one achieved by the random baseline. Both precision and recall values show notable improvements compared to the baseline method, with a F1 score of about 0.66.

The highest performance is achieved by the most advanced LLM, the *gpt-4* model. The results show an important reduction in SHD to an average of 1.6 errors, while precision rises to 0.89 and recall achieves an impressive 0.98, leading to a F1 score of 0.93. This score nearly triples the F1 score achieved by the random baseline, showing the non-trivial contribution of LLM outputs in facilitating the identification of causal graphs.

Model	SHD	Precision	Recall	F1 score
Random	8.3	0.334	0.379	0.355
gpt-3.5-turbo	4.166	0.710	0.622	0.663
gpt-4	**1.666**	**0.888**	**0.976**	**0.930**

Tabella 6.1: Benchmark results.

6.2 Causal Discovery from Real Medical Abstracts

The previously presented tests used existing causal graphs as ground-truth benchmarks.
It is thought important to test this approach within real-world contexts. Causal graph identification is a challenging task, especially when the text is complex and contains a lot of technical jargon. As previously discussed, medical texts are a particularly difficult case, as they are often full of abbreviations, acronyms, and other specialized terms. This makes it difficult for language models to identify the main entities in the text and to build accurate causal graphs. For this reason, the process was evaluated by extending its application to real-world scenarios, using actual medical texts.
As expected, the anticipated accuracy of the causal graphs tends to decrease as the complexity of the text increases. The results suggest that this approach is a promising one, but that there is still room for improvement.
LLMs are a powerful tool, but they are not perfect. They can be biased, make errors, and even hallucinate, by answering questions about obscure topics and making things up that sound plausible but are not actually true. Therefore, it is important to use them in conjunction with other methods, such as human judgment and domain knowledge.
The results of LLMs used for causal discovery can serve as a valuable head start, as they can be used to identify potential causal relationships by distinguishing words and phrases that are often used together in sentences that describe such relationships [7].
The following section presents a full example of the causal analysis process applied to the abstract of a real medical article extracted from the PubMed repository.

6.3 Example with Real Medical Text

The article used for the test was sourced from PubMed; the title is *"Research progress on the protective mechanism of a novel soluble epoxide hydrolase inhibitor TPPU on ischemic stroke"* [33] and the abstract is:

Arachidonic Acid (AA) is the precursor of cerebrovascular active substances in the human body, and its metabolites are closely associated with the pathogenesis of cerebrovascular diseases. In recent years, the cytochrome P450 (CYP) metabolic pathway of AA has become a research hotspot. Furthermore, the CYP metabolic pathway of AA is regulated by soluble epoxide *hydrolase (sEH). 1-trifluoromethoxyphenyl-3(1-propionylpiperidin-4-yl) urea (TPPU) is a novel sEH inhibitor that exerts cerebrovascular protective activity. This article reviews the mechanism of TPPU's protective effect on ischemic stroke disease.*

The extracted entities from the text are:

[*'Arachidonic Acid (AA)', 'cerebrovascular active substances', 'pathogenesis of cerebrovascular diseases', 'cytochrome P450 (CYP) metabolic pathway', 'soluble epoxide hydrolase (sEH)', TPPU'*].

In the original abstract, these are:

Arachidonic Acid (AA) is the precursor of cerebrovascular active substances in the human body, and its metabolites are closely associated with the pathogenesis of cerebrovascular diseases. In recent years, the cytochrome P450 (CYP) metabolic pathway of AA has become a research hotspot. Furthermore, the CYP metabolic pathway of AA is regulated by soluble epoxide hydrolase (sEH). 1-trifluoromethoxyphenyl-3(1-propionylpiperidin-4-yl) urea (TPPU) is a novel sEH inhibitor that exerts cerebrovascular protective activity. This article reviews the mechanism of TPPU's protective effect on ischemic stroke disease.

The process then queries the GPT LLM to find the causal relationships between the variables, as previously discussed. The resulting relationships are encoded in the corresponding causal graph, as shown in Figure 6.1.

6.4 Limitations

LLMs are very powerful tools. Despite their impressive potential, it's crucial to recognize the limitations they introduce in the context of causal analysis and the identification of causal graphs.
LLMs heavily depend on the textual data they are trained on, much of which is sourced from internet uploads. As a result, their performance can be influenced by biases and inaccuracies in the training data. Additionally, the language commonly used across the broader internet often features casual and colloquial language patterns that include causal terminology used in a way that differs from the precise and formal language found in medical academic literature. This can introduce further bias and lead the LLM to produce inaccurate or misleading answers. Furthermore, the complex nature of causal relationships, particularly

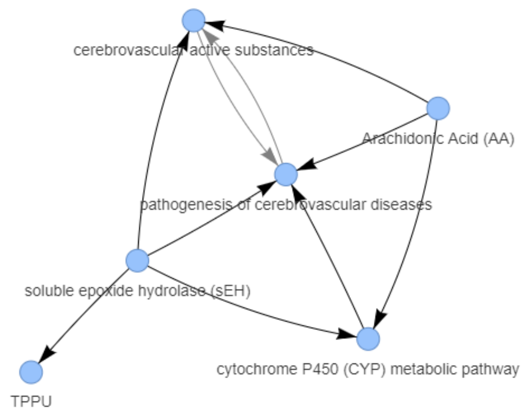

cerebrovascular active substances
Arachidonic Acid (AA)
pathogenesis of cerebrovascular diseases
soluble epoxide hydrolase (sEH)
cytochrome P450 (CYP) metabolic pathway
TPPU

Figura 6.1: Causal graph from real text example.

in domains with intricate interdependencies like the medical field, may present challenges for LLMs to consistently capture the nuanced dynamics of causation [2].
LLMs can also be computationally expensive, resulting in potentially slow performance. This can be a limitation in some applications that demand real-time identification of causal graphs. When testing the process with actual medical abstracts, particularly those involving very large texts, the pipeline could take several hours to complete. The operational bottleneck was observed to be the causal query step, during which the LLM was queried about the causal relationship between all pairs of variables. Depending on the text's size and the resulting number of extracted entities, the volume of queries could extend to thousands of GPT requests. Given an average performance of several seconds per GPT API request, the cumulative execution time for the entire pipeline process could extend to hours.
By addressing these limitations and aligning our expectations with the LLMs capabilities, researchers can make informed decisions about using them as valuable tools in the pursuit of identifying intricate cause-and-effect relationships within complex textual data.

Capitolo 7

Conclusion

This conclusion chapter summarizes the main findings of the study and discusses their implications. It also highlights the limitations of the study and suggests directions for future work.

7.1 Project Summary

This project investigated the capabilities of large language models, with a specific focus on their abilities in causal discovery and the identification of causal relationships.
The project's results, which show an average F1 score of 0.93 on the benchmarks, suggest that LLMs have the potential to significantly assist human efforts and contribute to tasks such as the identification of causal graphs.
The results also reveal that LLMs are not exempt from flaws or weaknesses. These imperfections highlight that the output and work of LLMs for building casual graphs should be verified by experts at this time. LLMs can be useful in extracting common knowledge from medical text, and when combined with expert insights, they offer the potential to efficiently generate more comprehensive causal graphs.
Large language models represent a remarkable advancement in artificial intelligence research. They are getting closer to human-level language capabilities than ever before [8]. LLMs represent a new and intriguing opportunity to extract common knowledge from medical literature and can help to complement and speed up causal analysis and causal graph identification. However, more research is essential to address and overcome the limitations of LLMs.

7.2 Future Work

Since this project focused primarily on identifying causal graphs, future efforts could prioritize the development of additional functionalities within the main causal analysis pipeline. For

example, the process could ensure the acyclicity of the generated graphs, another important property of DAGs. This might involve developing methods to detect and rectify incorrect or misplaced causal relationships (graph edges).
Another potential feature could involve labeling causal relationships (in the form of graph edges) with citations to the sources that the LLM referenced to establish the direction of causality between specific pairs of entities. This would aide human experts in assessing the validity and accuracy of LLM-generated answers by verifying the information against reliable and legitimate sources.
The findings and results of this project suggest several potential avenues for future development and research. As the field of large language models continues to evolve, exploring how these models can be further fine-tuned and tailored to domain-specific contexts could achieve even more accurate and robust causal analysis results. Investigating ways to mitigate the identified limitations of LLMs, such as improving the accuracy of extracting nuanced causal relationships or refining the verification process by experts, would help to maximize the potential of LLMs in causal graph identification.

Bibliografia

[1] S. Long et al. “Causal Discovery with Language Models as Imperfect Experts”. In: *arXiv e-prints*, arXiv:2307.02390 (lug. 2023), arXiv:2307.02390. DOI: 10.48550/arXiv.2307.02390. arXiv: 2307.02390 [cs.AI].

[2] S. Long et al. “Can large language models build causal graphs?” In: *arXiv e-prints*, arXiv:2303.05279 (mar. 2023), arXiv:2303.05279. DOI: 10.48550/arXiv.2303.05279. arXiv: 2303.05279 [cs.CL].

[3] J. Pearl. “Theoretical Impediments to Machine Learning With Seven Sparks from the Causal Revolution”. In: (2018). DOI: 10.1145/3159652.3176182.

[4] E. Kıcıman et al. “Causal Reasoning and Large Language Models: Opening a New Frontier for Causality”. In: *arXiv e-prints*, arXiv:2305.00050 (apr. 2023), arXiv:2305.00050. DOI: 10.48550/arXiv.2305.00050. arXiv: 2305.00050 [cs.AI].

[5] C. Glymour, K. Zhang e P. Spirtes. “Review of Causal Discovery Methods Based on Graphical Models”. In: *Frontiers in Genetics* 10 (2019). ISSN: 1664-8021. DOI: 10.3389/fgene.2019.00524. URL: https://www.frontiersin.org/articles/10.3389/fgene.2019.00524.

[6] T. Verma e J. Pearl. “Equivalence and Synthesis of Causal Models”. In: *Probabilistic and Causal Inference* (1990). URL: https://api.semanticscholar.org/CorpusID:27807863.

[7] M. Willig et al. “Causal Parrots: Large Language Models May Talk Causality But Are Not Causal”. In: *Submitted to Transactions on Machine Learning Research* (2023). Under review. URL: https://openreview.net/forum?id=tv46tCzs83.

[8] C. Zhang et al. “Understanding Causality with Large Language Models: Feasibility and Opportunities”. In: *arXiv e-prints*, arXiv:2304.05524 (apr. 2023), arXiv:2304.05524. DOI: 10.48550/arXiv.2304.05524. arXiv: 2304.05524 [cs.LG].

[9] A. Zanga e F. Stella. “A Survey on Causal Discovery: Theory and Practice”. In: *arXiv e-prints*, arXiv:2305.10032 (mag. 2023), arXiv:2305.10032. DOI: 10.48550/arXiv.2305.10032. arXiv: 2305.10032 [cs.AI].

[10] *A COMPLETE GUIDE TO Natural Language Processing*. URL: https://www.deeplearning.ai/resources/natural-language-processing/. (accessed: 23.08.2023).

[11] *spaCy · Industrial-strength Natural Language Processing in Python*. URL: https://spacy.io/. (accessed: 23.08.2023).

[12] *flair*. URL: https://flairnlp.github.io/. (accessed: 23.08.2023).

[13] J. Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv e-prints*, arXiv:1810.04805 (ott. 2018), arXiv:1810.04805. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805 [cs.CL].

[14] *API Reference - OpenAI API*. URL: https://platform.openai.com/docs/api-reference/introduction?lang=python. (accessed: 23.08.2023).

[15] *GPT - OpenAI API*. URL: https://platform.openai.com/docs/guides/gpt. (accessed: 29.06.2023).

[16] I. Fulford e A. Ng. *ChatGPT Prompt Engineering for Developers*. URL: https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/. (accessed: 10.08.2023).

[17] *Prompt Engineering Guide*. URL: https://www.promptingguide.ai/. (accessed: 30.06.2023).

[18] M. Hobbhahn e T. Lieberum. *Investigating causal understanding in LLMS*. URL: https://www.lesswrong.com/posts/yZb5eFvDoaqB337X5/investigating-causal-understanding-in-llms. (accessed: 02.06.2023).

[19] The PyCoach. *OpenAI and Andrew Ng Just Released a FREE ChatGPT Prompt Engineering Course*. URL: https://artificialcorner.com/openai-and-andrew-ng-just-released-a-free-chatgpt-prompt-engineering-course-b0884c03e946. (accessed: 10.08.2023).

[20] J. Shieh. *Best practices for prompt engineering with OpenAI API*. URL: https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api. (accessed: 10.08.2023).

[21] *PubMed - Wikipedia*. URL: https://en.wikipedia.org/wiki/PubMed. (accessed: 30.06.2023).

[22] E. Sayers. *A General Introduction to the E-utilities*. URL: https://www.ncbi.nlm.nih.gov/books/NBK25497/. (accessed: 02.06.2023).

[23] T. P. Peixoto. “The graph-tool python library”. In: *figshare* (2014). DOI: 10.6084/m9.figshare.1164194. URL: http://figshare.com/articles/graph_tool/1164194 (visitato il 10/09/2014).

[24] Robert Tarjan. “Depth-First Search and Linear Graph Algorithms”. In: *SIAM Journal on Computing* 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010. eprint: https://doi.org/10.1137/0201010. URL: https://doi.org/10.1137/0201010.

[25] D. Johnson. “Finding All the Elementary Circuits of a Directed Graph”. In: *SIAM Journal on Computing* 4.1 (1975), pp. 77–84. DOI: 10.1137/0204007. eprint: https://doi.org/10.1137/0204007. URL: https://doi.org/10.1137/0204007.

[26] A. A. Hagberg, D. A. Schult e P. J. Swart. “Exploring Network Structure, Dynamics, and Function using NetworkX”. In: *Proceedings of the 7th Python in Science Conference*. A cura di Gaël Varoquaux, Travis Vaught e Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[27] Khuyen T. *Pyvis: Visualize Interactive Network Graphs in Python*. URL: https://towardsdatascience.com/pyvis-visualize-interactive-network-graphs-in-python-77e059791f01. (accessed: 01.07.2023).

[28] J. M. Mooij et al. “Distinguishing cause from effect using observational data: methods and benchmarks”. In: *arXiv e-prints*, arXiv:1412.3773 (dic. 2014), arXiv:1412.3773. DOI: 10.48550/arXiv.1412.3773. arXiv: 1412.3773 [cs.LG].

[29] R. Tu, C. Ma e C. Zhang. “Causal-Discovery Performance of ChatGPT in the context of Neuropathic Pain Diagnosis”. In: *arXiv e-prints*, arXiv:2301.13819 (gen. 2023), arXiv:2301.13819. DOI: 10.48550/arXiv.2301.13819. arXiv: 2301.13819 [cs.CL].

[30] M. Scutari. *Bayesian Network Repository*. URL: https://www.bnlearn.com/bnrepository/. (accessed: 20.07.2023).

[31] D. Kalainathan e O. Goudet. “Causal Discovery Toolbox: Uncover causal relationships in Python”. In: *arXiv e-prints*, arXiv:1903.02278 (mar. 2019), arXiv:1903.02278. DOI: 10.48550/arXiv.1903.02278. arXiv: 1903.02278 [stat.CO].

[32] A. Nogueira et al. “Methods and tools for causal discovery and causal inference”. In: *WIREs Data Mining and Knowledge Discovery* 12.2 (2022). DOI: https://doi.org/10.1002/widm.1449. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1449. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1449.

[33] H. Pan. “Research progress on the protective mechanism of a novel soluble epoxide hydrolase inhibitor TPPU on ischemic stroke”. In: *Frontiers in Neurology* 14 (2023). ISSN: 1664-2295. DOI: 10.3389/fneur.2023.1083972. URL: https://www.frontiersin.org/articles/10.3389/fneur.2023.1083972.

Appendix

Appendice A

Implementation Details

This appendix section presents the source code used for this project.

A.1 Scraping

Search PubMed articles by terms

```python
def search_by_terms(terms, db, retmax, use_history):

    terms_string = '+AND+'.join([s.strip().replace(' ', '+') for s in terms])
    url_params = {
            'db': db,
            'term': terms_string,
            'retmax': retmax,
            'api_key': api_key,
        }

    if use_history:
        url_params['usehistory'] = 'y'

    url = f'{base_url}esearch.fcgi'
    response = requests.get(url, params=url_params)
    ids = re.findall(r"<Id>(\d+)</Id>", response.text)

    if use_history:
        web_match = re.search(r"<WebEnv>(\S+)</WebEnv>", response.text)
        web = web_match.group(1) if web_match else None

        key_match = re.search(r"<QueryKey>(\d+)</QueryKey>", response.text)
        key = key_match.group(1) if key_match else None

        return ids, web, key

    return ids
```

Get data of previously found articles

```python
def get_articles_data(ids, web_env, query_key, db, retmax):

    use_web_env = not ids

    url_params = {
        'db': db,
        'rettype': 'abstract',
        'retmode': 'xml',
        'api_key': api_key,
        'retmax': retmax,
    }

    if use_web_env:
        url_params['query_key'] = query_key
        url_params['WebEnv'] = web_env
    else:
        ids_string = ','.join(map(str, ids))
        url_params['id'] = ids_string

    url = f'{base_url}efetch.fcgi'
    response = requests.get(url, params=url_params)

    soup = BeautifulSoup(response.text, features="xml")
    articles = soup.find_all('PubmedArticle')
    if not articles:
        print('ERROR: No articles found')
        return None

    data = pd.DataFrame(columns=['id', 'title', 'abstract', 'keywords', '
pub_date'])
    for article in articles:
        article_data = {
            'id': article.find('PMID').get_text(),
            'title': article.find('ArticleTitle').get_text(),
            'abstract': ' '.join([a.get_text() for a in article.find_all(
'AbstractText')]),
            'keywords': [[k.get_text() for k in article.find_all('Keyword
')]],
        }
        pub_date = article.find('PubMedPubDate', {'PubStatus': 'received'
})
        if pub_date:
            article_data['pub_date'] = datetime.strptime(f"{pub_date.find('Day').get_text()} {pub_date.find('Month').get_text()} {pub_date.find('Year').get_text()}", "%d %m %Y")

        data = pd.concat([data, pd.DataFrame(article_data)]).reset_index(drop=True)

    return data
```

Clean invalid data

```
def clean_data(data, drop_id_duplicates, drop_empty_abstracts, drop_nan_abstracts, drop_date_nan, drop_date_before, drop_date_after, search_terms):

    if drop_id_duplicates:
        data = data.drop_duplicates(subset=['id'], inplace=False)
    if drop_empty_abstracts:
        data = data[data['abstract'] != '']
    if drop_nan_abstracts:
        data = data.dropna(subset=['abstract'])
    if drop_abstracts_with_matches and drop_abstracts_matches:
        data = data[~data['abstract'].str.startswith(tuple(drop_abstracts_matches))]

    if drop_date_nan:
        data = data.dropna(subset=['pub_date'])

    if drop_date_before:
        data = data[data['pub_date'] > drop_date_before]
    if drop_date_after:
        data = data[data['pub_date'] < drop_date_after]

    if search_terms:
        data['search_terms'] = [search_terms]*len(data)

    return data.reset_index(drop=True)
```

A.2 Causal Discovery

The complete GPT user messages can be found at Appendix B.

Complete *causal_discovery_pipeline*

```python
def causal_discovery_pipeline(text_title, text, entities,
    reverse_edge_for_variable_check, optimize_found_entities,
    search_cycles):

    if entities == []:
        entities = gpt_ner(text)

    if optimize_found_entities:
        opt_entities = optimize_entities(entities, text)
        entities = list(opt_entities.keys())

    graph_edges = gpt_causal_discovery(entities, text)

    edges = extract_edge_answers(graph_edges)

    if reverse_edge_for_variable_check:
        valid_edges, invalid_edges = check_invalid_answers(edges)

        edge_correction_response = correct_invalid_edges(invalid_edges,
    text)
        corrected_edges = extract_edge_answers(edge_correction_response)

        valid_edges.extend(corrected_edges)
        edges = valid_edges

    nodes, processed_edges, bidirected_edges = preprocess_edges(edges)

    cycles = []
    if search_cycles:
        cycles = find_cycles(nodes=nodes, edges=processed_edges)
    build_graph(nodes, processed_edges, bidirected_edges, cycles)


    if verbose:
        print_edges(graph_edges)

    return nodes, processed_edges + bidirected_edges, cycles
```

Base GPT API request helper function

```python
def gpt_request(system_msg, user_msg, model, temperature):
    if not system_msg or not user_msg:
        return None
    try:
        response = openai.ChatCompletion.create(
                            model=model,
                            messages=[
                                  {"role": "system", "content": system_msg},
                                  {"role": "user", "content": user_msg}],
                                 temperature=temperature)

        return response.choices[0].message.content
    except:
        return None
```

Entity extraction function

```
def gpt_ner(text):

    system_msg = 'You are a helpful assistant for Named Entity
   Recognition of medical texts.'

    # see Appendix B.1 for user message
    # user_msg = ''

    response = gpt_request(system_msg, user_msg)
    if not response:
        return []

    answer_text = response

    soup = BeautifulSoup(answer_text, 'html.parser')
    entities = [entity.text for entity in soup.find_all('entity')]

    return entities
```

Entity optimization function

```
def optimize_entities(entities, text=None):
    system_msg = 'You are a helpful assistant for medical entity
   optimization, by accurately identifying synonyms, redundant entities,
   or entities that can be used interchangeably'

    entities_text = '\n'.join([f'<Entity>{entity}</Entity>' for entity in
    entities])

    # see Appendix B.2 for user message
    # user_msg = ''

    response = gpt_request(system_msg, user_msg)
    if response:
        soup = BeautifulSoup(response, 'html.parser')
        answer = soup.find('answer').text
        try:
            opt_entities = json.loads(answer)
            if opt_entities:
                return opt_entities
        except (json.JSONDecodeError, TypeError):
            pass

    return entities
```

Causal discovery function

```
def gpt_causal_discovery(entities, text, use_pretrained_knowledge,
    reverse_variable_check):

    graph_edges = []

    system_msg = 'You are a helpful assistant for causal reasoning and
    cause-and-effect relationship discovery.'

    for i1, e1 in enumerate(entities):
        for i2, e2 in enumerate(entities):
            if i1 == i2 or (not reverse_variable_check and i1 >= i2):
                continue


            # see Appendix B.3 for user message
            # user_msg = ''

            response = gpt_request(system_msg, user_msg)
            if response:
                graph_edges.append(((e1, e2), response))

    return graph_edges
```

Compatibility check for GPT answers

```
def check_edge_compatibility(answer1, answer2):
    return (arrows[answer1], arrows[answer2]) in [(forward_arrow,
    backward_arrow), (backward_arrow, forward_arrow), (no_arrow, no_arrow)
    , (bidirectional_arrow, bidirectional_arrow)]
```

Find invalid GPT answers, when using double variable check

```
def check_invalid_answers(directed_edges):
    invalid_edges = []
```

```python
    valid_edges = []
    temp_edges = []
    answers = {}
    for (n1, n2), answer in directed_edges:

        if (n1, n2) not in temp_edges and (n2, n1) not in temp_edges:
            temp_edges.append((n1, n2))
            answers[(n1, n2)] = answer
        elif (n1, n2) in temp_edges:
            if answers[(n1, n2)] != answer:
                invalid_edges.append((((n1, n2), answer), ((n2, n1), answers[(n2, n1)])))
            else:
                valid_edges.append(((n1, n2), answer))

            temp_edges.remove((n1, n2))
        elif (n2, n1) in temp_edges:
            if check_edge_compatibility(answers[(n2, n1)], answer):
                valid_edges.append(((n1, n2), answer))
            else:
                invalid_edges.append((((n1, n2), answer), ((n2, n1), answers[(n2, n1)])))

            temp_edges.remove((n2, n1))

    for n1, n2 in temp_edges:
        if (n1, n2) not in invalid_edges:
            invalid_edges.append((((n1, n2), answer), ((n2, n1), answers[(n2, n1)])))

    return valid_edges, invalid_edges
```

Return GPT answers in textual form, when re-querying for incoherent answers

```python
def get_textual_answers(e1, e2, ans):
    if ans == forward_arrow_answer:
        return f'"{e1}" causes "{e2}"'
    elif ans == backward_arrow_answer:
        return f'"{e2}" causes "{e1}"'
    elif ans == no_arrow_answer:
        return f'"{e1}" and "{e2}" are not causally related'
    elif ans == bidirectional_arrow_answer:
        return f'there is a common factor that is the cause for both "{e1}" and "{e2}"'
    else:
        return None
```

Correct incoherent GPT answers

```python
def correct_invalid_edges(invalid_edges, text, use_pretrained_knowledge):
    graph_edges = []

    if not invalid_edges:
        return []

    system_msg = 'You are a helpful assistant for causal reasoning and cause-and-effect relationship discovery.'

    for ((e1, e2), answer1), ((e3, e4), answer2) in invalid_edges:

        # see Appendix B.4 for user message
        # user_msg = ''

        response = gpt_request(system_msg, user_msg)
        if response:
            graph_edges.append(((e1, e2), response))

    return graph_edges
```

A.3 Plotting the Causal Graph

Decode GPT answers in directed edges

```python
def preprocess_edges(edges):
    nodes = set()
    directed_edges = []
    bidirected_edges = []

    for (n1, n2), answer in edges:

        nodes.add(n1)
        nodes.add(n2)

        direction = normalize_edge_direction(n1, n2, answer)
        if direction:
            if len(direction) == 2:
                bidirected_edges.extend(direction)
            else:
                directed_edges.extend(direction)

    return list(nodes), directed_edges, bidirected_edges
```

Return edge in normalized form (A → B)

```python
def normalize_edge_direction(e1, e2, answer):
    if answer in arrows:
        if arrows[answer] == forward_arrow:
            return [(e1, e2)]
        elif arrows[answer] == backward_arrow:
            return [(e2, e1)]
        elif arrows[answer] == bidirectional_arrow:
            return [(e2, e1), (e1, e2)]
    return None
```

Find cycles in causal graph

```python
def find_cycles(nodes, edges):
    if not nodes or not edges:
        return []

    g = gt.Graph(directed=True)

    nodes_ids = {}
    v_prop = g.new_vertex_property("string")
    for n in nodes:
        v = g.add_vertex()
        v_prop[v] = n
        nodes_ids[n] = v

    for (n1, n2) in edges:
        e = g.add_edge(nodes_ids[n1], nodes_ids[n2])

    cycles = []
    for i, c in enumerate(gt.all_circuits(g)):
        if i >= 100:
            break
        cycles.append([v_prop[v] for v in c])

    return cycles
```

Build resulting causal graph

```python
def build_graph(nodes, edges, bidirected_edges, cycles, plot_static_graph
    , directory_name, graph_name):

    if plot_static_graph:
        plt.figure()
    G = nx.DiGraph()

    G.add_nodes_from(nodes)

    for e1, e2 in edges:
        G.add_edge(e1, e2, color='black', style='solid')

    for cycle in cycles:
        for i in range(len(cycle) - 1):
            G[cycle[i]][cycle[i + 1]]['color'] = 'red'
        G[cycle[-1]][cycle[0]]['color'] = 'red'

    for e1, e2 in bidirected_edges:
        G.add_edge(e1, e2, color='grey', style='dashed')

    if plot_static_graph:
        pos = nx.spring_layout(G)
        nx.draw_networkx_nodes(G, pos)
        nx.draw_networkx_labels(G, pos)

        edge_colors = [G.edges[edge]['color'] for edge in G.edges()]
        edge_styles = [G.edges[edge]['style'] for edge in G.edges()]

        nx.draw(G, pos, node_color='skyblue', node_size=1500,
                font_size=10, font_weight='bold', arrowsize=20,
   edge_color=edge_colors, style=edge_styles,
                width=2)
        plt.title(graph_name)
        plt.show()

    net = Network(directed=True, notebook=True)
    net.from_nx(G)
    net.force_atlas_2based()
    net.show_buttons(filter_=['physics'])
    os.makedirs(directory_name, exist_ok=True)
    net.save_graph(f'{directory_name}/{graph_name}.html')
```

Appendice B

GPT User Messages

The following section contains examples of all user messages used within the project. These were constructed with an iterative prompt development approach [16].

B.1 Named Entity Recognition (NER)

```
You will be provided with an abstract of a medical research paper delimited by the <Text></Text> xml tags.
Please read the provided abstract carefully to comprehend the context and content. Analyze the provided text and identify all the meaningful entities that could contribute to understanding cause-and-effect relationships between factors such as diseases, medications, treatments, interventions, symptoms, outcomes, effects, or risk factors.

Avoid including entities that are synonyms or can be used interchangeably to already identified ones. For example, if text contains both "hospital" and "medical center" (which are synonyms and can be used interchangeably) and you already extracted "hospital" as final entity, do not include "medical center" as well.

Your response should highlight entities that are crucial for establishing causal relationships in the medical context.

Answer listing only the found entities within the tags <Answer><Entity>[entity]</Entity><Entity>[entity]</Entity></Answer>
(e.g., <Answer><Entity>diabetes</Entity><Entity>hypertension</Entity></Answer>)

Text:
    <Text>{text}</Text>
```

B.2 Entity Optimization

```
You will be provided with an abstract of a medical research paper delimited by the <Text></Text> xml tags, and a list of named entities representing medical entities, each one of them delimited by the <Entity></Entity> xml tags.

Text:
        <Text>{text}</Text>

Entities:
        {entities}

Your task is to optimize this entity list by identifying synonyms within the entities and grouping them accordingly.
Your goal is to create a JSON object where the keys represent the root word entities, and each key is associated with an array of its synonyms, i.e., words or entities that can be used interchangeably to the root word.
If a root word entity has no synonyms, its value in the JSON should be an empty array.

Ensure that each entity appears only once in the dictionary, either as key (i.e. root word) or as element in the value arrays (i.e. the synonyms): an entity must not appear as key if it is the synonym (i.e. in the value array) of another entity, and the other way around (i.e. must not be in the value array of an entity if it is already a key of the JSON object).
An entity must not be a synonym of itself.

You should efficiently process the given list of entities and produce the desired dictionary structure.
The output JSON object should accurately represent the optimized entities and their synonyms based on the provided list.

Then provide your final JSON object answer within the tags <Answer>[json_obj]</Answer>, (e.g. <Answer>
    {{
        "smoking": ["tobacco", "nicotine", "cigarettes", "cigar"],
        "cancer": ["lung cancer"],
        "tumors": []
     }}
     </Answer>
).

Follow the example below to understand the expected output.

```

```
 Example:

 Given the initial list of entities:
 <Entity>smoking</Entity>
 <Entity>lung cancer</Entity>
 <Entity>tumors</Entity>
 <Entity>cancer</Entity>
 <Entity>tobacco</Entity>
 <Entity>nicotine</Entity>
 <Entity>cigarettes</Entity>
 <Entity>cigar</Entity>

 You should pair the synonyms, generate the following JSON object and
provide it as your answer:
 <Answer>
 {{
     "smoking": ["tobacco", "nicotine", "cigarettes", "cigar"],
     "cancer": ["lung cancer"],
     "tumors": []
 }}
 </Answer>

 Note that every entity appears only once in the output JSON object,
either as key or as element in the value arrays.

 After you have finished building the JSON object, check and make sure
  that every entity appears only once in the output JSON object, either
  as key or as element in the value arrays.
```

B.3 Causal Discovery

The *random_causal_verb* variable is randomly selected from the available causation verbs, which include: ’provokes’, ’triggers’, ’causes’, ’leads to’, ’induces’, ’results in’, ’generates’, ’produces’, ’stimulates’, ’instigates’, ’engenders’, ’promotes’, ’gives rise to’, ’sparks’ [1].

```
 You will be provided with an abstract of a medical research paper
delimited by the <Text></Text> xml tags, and a pair of entities
extracted from the given abstract delimited by the <Entity></Entity>
xml tags representing medical entities, such as medications,
treatments, symptoms, diseases, outcomes, side effects, or other
medical factors.

     Text:
             <Text>{text}</Text>

     Entities:
             <Entity>{entity1}</Entity>
```

```
                <Entity>{entity2}</Entity>

    Please read the provided abstract carefully to comprehend the context
     and content. Examine the roles, interactions, and details surrounding
     the entities within the abstract.
    Based on the information in the text and your pretrained knowledge,
    determine the most likely cause-and-effect relationship between the
    entities from the following listed options (A, B, C, D):

        Options:
                A: "{entity1}" {random_causal_verb} "{entity2}";
                B: "{entity2}" {random_causal_verb} "{entity1}";
                C: "{entity1}" and "{entity2}" are not directly causally
    related;
                D: there is a common factor that is the cause for both "{
    entity1}" and "{entity2}";

    Your response should analyze the situation in a step-by-step manner,
    ensuring the correctness of the ultimate conclusion, which should
    accurately reflect the likely causal connection between the two
    entities based on the information presented in the text and any
    additional knowledge you are aware of.
    If no clear causal relationship is apparent, select the appropriate
    option accordingly.

    Then provide your final answer within the tags <Answer>[answer]</
    Answer>, (e.g. <Answer>C</Answer>).
```

B.4 Correct Incoherent GPT Answers

The *get_textual_answers* function returns a textual version of the causal relationship between the two entities, in the form of "*'A' causes 'B'*". For *random_causal_verb* see Appendix B.3.

```
    You will be provided with an abstract of a medical research paper
    delimited by the <Text></Text> xml tags, and a pair of entities
    extracted from the given abstract delimited by the <Entity></Entity>
    xml tags representing medical entities (such as medications,
    treatments, symptoms, diseases, outcomes, side effects, or other
    medical factors), and two answers you previously gave to this same
    request that are incoherent with each other, delimited by the <Answer
    ></Answer> xml tags.
        Text:
                <Text>{text}</Text>

        Entities:
                <Entity>{entity1}</Entity>
                <Entity>{entity2}</Entity>

        Previous incoherent answers:
                <Answer>{get_textual_answers(entity1, entity2, answer1)}</Answer>
                <Answer>{get_textual_answers(entity2, entity1, answer2)}</Answer>

Please read the provided abstract carefully to comprehend the context and content.
Consider the previous answers you gave to this same request that are incoherent with each other, and the entities they refer to in order to give a correct answer. Examine the roles, interactions, and details surrounding the entities within the abstract.
Based on the information in the text and your pretrained knowledge, determine the most likely cause-and-effect relationship between the entities from the following listed options (A, B, C, D):

        Options:
            A: "{entity1}" {random_causal_verb} "{entity2}";
            B: "{entity2}" {random_causal_verb} "{entity1}";
            C: "{entity1}" and "{entity2}" are not directly causally related;
            D: there is a common factor that is the cause for both "{entity1}" and "{entity2}";

Your response should accurately reflect the likely causal connection between the two entities based on the information presented in the text and any additional knowledge you are aware of.
If no clear causal relationship is apparent, select the appropriate option accordingly.
Then provide your final answer within the tags <Answer>[answer]</Answer>
(e.g. <Answer>C</Answer>).
```

Appendice C

Benchmarks

This appendix section presents the benchmarks in more detail. It shows the ground truth graphs and, using the evaluation metrics discussed at section 5.3, presents the additional results from the different algorithms: the random baseline, the *gpt-3.5-turbo* and *gpt-4* models.

C.1 Ground Truth Graphs

The ground truth graphs for the benchmarks are shown at Figures 5.1, C.1, C.2, C.3, C.4, C.5.

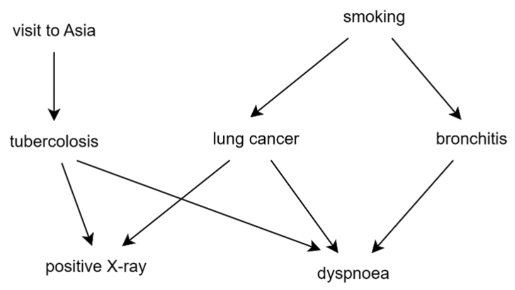


Figura C.1: ’Asia’ benchmark.

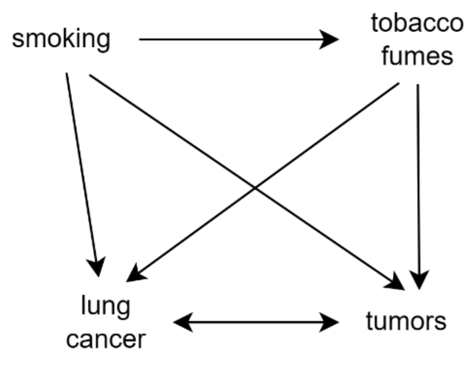


Figura C.2: ’Smoking’ benchmark.

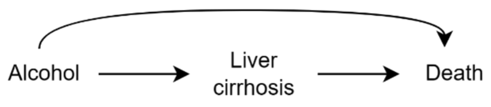


Figura C.3: ’Alcohol’ benchmark.

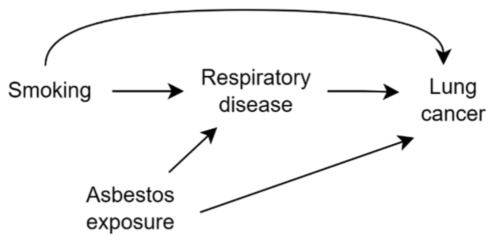


Figura C.4: ’Cancer’ benchmark.

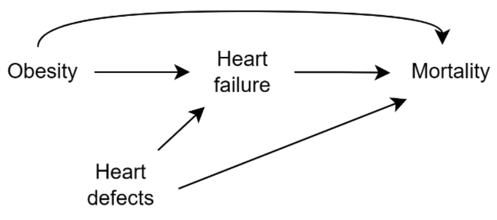


Figura C.5: ’Obesity’ benchmark.

C.2 Additional Benchmark Results

The results of the various algorithms applied to the benchmarks are presented in Tables C.1 (baseline), C.2 (*gpt-3.5-turbo*), and C.3 (*gpt-4*).

Benchmark Name	Missing edges	Extra edges	SHD	Correct edge direction	Incorrect edge direction	Precision	Recall	F1 score	PRC Area
Asia	6	15	21	2	15	0.118	0.25	0.16	0.245
Smoking	4	1	5	2	2	0.75	0.428	0.545	0.714
Alcohol	3	1	4	0	1	0	0	NaN	0.166
Cancer	2	4	6	2	5	0.428	0.6	0.5	0.576
Diabetes	2	5	7	1	7	0.375	0.6	0.461	0.55
Obesity	3	4	7	1	5	0.333	0.4	0.363	0.46
AVG	3.333	5	8.333	1.333	5.833	0.334	0.379	0.355	0.452

Tabella C.1: Benchmark results with random baseline.

Benchmark Name	Missing edges	Extra edges	SHD	Correct edge direction	Incorrect edge direction	Precision	Recall	F1 score	PRC Area
Asia	3	5	8	5	5	0.5	0.625	0.555	0.593
Smoking	3	0	3	4	0	1	0.571	0.727	0.879
Alcohol	2	0	2	1	0	1	0.333	0.5	0.778
Cancer	0	1	1	4	2	0.833	1	0.909	0.917
Diabetes	2	4	6	1	6	0.429	0.6	0.5	0.577
Obesity	2	3	5	2	4	0.5	0.6	0.545	0.612
AVG	2	2.166	4.166	2.833	2.833	0.710	0.622	0.663	0.726

Tabella C.2: Benchmark results with *gpt-3.5-turbo* model.

Benchmark Name	Missing edges	Extra edges	SHD	Correct edge direction	Incorrect edge direction	Precision	Recall	F1 score	PRC Area
Asia	0	8	8	8	8	0.5	1	0.666	0.75
Smoking	1	0	1	5	1	1	0.857	0.923	0.96
Alcohol	0	0	0	3	0	1	1	1	1
Cancer	0	1	1	4	2	0.8333	1	0.909	0.917
Diabetes	0	0	0	5	0	1	1	1	1
Obesity	0	0	0	5	0	1	1	1	1
AVG	0.166	1.5	1.666	5	1.833	0.888	0.976	0.930	0.937

Tabella C.3: Benchmark results with *gpt-4* model.

Appendice D

Command Line Options

This appendix section presents all command line options for running the whole process using the previously presented *causal_llms.py* script.

```
Usage: causal_llms.py <action> [options]

Description:
  This script performs various tasks related to causal discovery.

Actions:
  ex       Run the example test.
  s        Run the scraping process.
  c        Perform causal analysis.
  sc       Run scraping and causal analysis.
  b        Run the benchmark tests.

Options:
  --help         Show this help message and exit.
  --data-path    Path of data file for causal analysis.
  --algorithm    Algorithm to use with benchmarks.


Examples:
  python causal_llms.py ex
  python causal_llms.py s
  python causal_llms.py c --data-path </path/to/data>
  python causal_llms.py sc
  python causal_llms.py b --algorithm {b|gpt}


The `algorithm` parameter sets the algorithm for the benchmark tests.
The possible values are:
* `b`: Baseline algorithm
* `gpt`: GPT LLM
```