Causal Graph Identification by LLMs

Sommario

[1. Abstract 3](#_Toc143531927)

[2. Introduction 4](#_Toc143531928)

[2.1 Project Objective 4](#_Toc143531929)

[3. State of the art 5](#_Toc143531930)

[4. Methodology and Techniques 6](#_Toc143531931)

[4.1 Causal analysis 6](#_Toc143531932)

[4.2 Named Entity Recognition 8](#_Toc143531933)

[4.3 GPT API 8](#_Toc143531934)

[5. Implementation 11](#_Toc143531935)

[5.1 PubMed scraping 11](#_Toc143531936)

[5.2 Causal analysis 14](#_Toc143531937)

[5.3 Plotting the causal graph 18](#_Toc143531938)

[5.4 Running the process 19](#_Toc143531939)

[6. Benchmarks and Metrics 20](#_Toc143531940)

[6.1 Pairwise Causal Relationship Discovery 20](#_Toc143531941)

[6.2 Full Causal Graph Identification 20](#_Toc143531942)

[6.1 Evaluation metrics 20](#_Toc143531943)

[7. Results and Discussion 22](#_Toc143531944)

[7.1 Causal discovery from benchmark tests 22](#_Toc143531945)

[7.2 Causal discovery from real medical abstracts 22](#_Toc143531946)

[7.3 Limitations 23](#_Toc143531947)

[Appendix 25](#_Toc143531948)

[A. Implementation detail 25](#_Toc143531949)

[1. Scraping 25](#_Toc143531950)

[2. Causal discovery 27](#_Toc143531951)

[3. Plotting the causal graph 32](#_Toc143531952)

[B. GPT user messages 35](#_Toc143531953)

[C. Benchmarks 39](#_Toc143531954)

[D. Example with real medical text 42](#_Toc143531955)

[Sources: 44](#_Toc143531956)

1. 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 [8].  
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 aide 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, as shown by multiple recent research papers and publications [1] [5] [8] [18] [21].

1. Introduction

Advances in causal inference is vital 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. It is a fundamental way of understanding the world, and it can be used to build more accurate and reliable AI systems. Machines and systems with a model of reality, similar to that used in causality, could have the potential to achieve strong AI and artificial general intelligence (AGI) [5].

Another field where causality is vital is the medical one. In medicine, most of the asked research questions are not associational, 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 [5].

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 (LLMs), which have been trained on vast volumes of textual data [5].

* 1. Project Objective

The primary goal of this project is to conduct an empirical study to assess the possibility of performing causal analysis using Large Language Models (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 (DGPO) 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 inferring the effects of interventions, making predictions, and assigning credit [1].

1. State of the art

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 can be divided into two main classes: constraint-based methods and score-based methods [15].

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 maximizes the assigned score.

Both these methods include different algorithms and techniques. While these techniques 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 [1][13].

Different approaches and efforts have tried to overcome this limitation, 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 [1].

A different approach, known as knowledge-based method, focuses 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 [5].

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 [1].

1. Methodology and Techniques

This section focuses on presenting the methods, approaches and tools used in the project.

As previously stated, the project 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 directed acyclic graph (DAG).

* 1. Causal analysis

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 section presents a set of definitions [6] for the main concepts and assumptions on causality and graph theory.

**Definitions**

Definition 0: **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.

Definition 1: **Graph Theory**  
Graph theory is a branch of mathematics that deals with the study of 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 ⊆ V × V.

Definition 3: **Directed Graph**A directed graph (DG) G is a graph where the edge (X, Y) is distinct from the edge (Y, X).

Definition 4: **Path**  
A path π = (X − · · · − Y) is a tuple of non-repeating vertices, where each vertex is connected to the next in the sequence with an edge.

Definition 5: **Directed Path**A directed path π = (X → · · · → Y) is a tuple of non-repeating vertices, where each vertex is connected to the next in the sequence with a directed edge.

Definition 6: **Cycle**  
A cycle is a path that starts and ends at the same vertex.

Definition 7: **Directed Acyclic Graph**A directed acyclic graph (DAG) is a directed graph G that has no cycles.

Definition 8: **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 (DAGs), encode contextual knowledge of variables (both observable and unobservable) and their causal dependency.

A crucial property of 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.

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

* The **directed edge** (*A 🡪 B*) denotes a direct causal dependence 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 <-> B* or both *A 🡪 B* and *A 🡨 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 10: **Direct and Indirect Cause**For each directed edge (X, Y) ∈ 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 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".

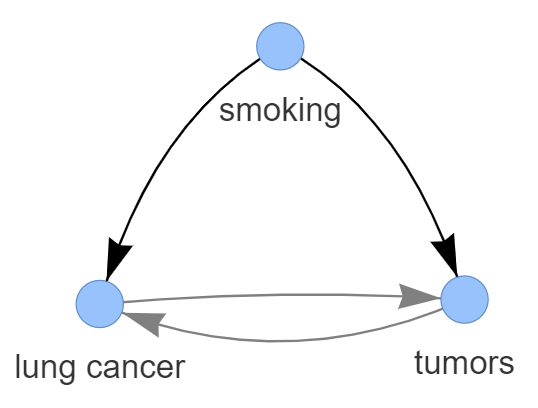


Figure 1: Causal graph example

* 1. Named Entity Recognition

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. The data used for constructing these causal graphs is collected in a textual form, from abstracts of medical publications and research papers.

Using the Named Entity Recognition operation, the key elements of the text can be identified and isolated.

Named Entity Recognition (NER) is a crucial natural language processing (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.

* 1. GPT API

Part of the project's activities, such as causal relationship identification and the NER operation, were performed using the GPT LLM through the GPT API.

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 works by sending requests to the API endpoint with a given prompt, and in return, it generates context-aware text based on the provided input.

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 the generated text as a response.

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 [2].

**Using the GPT API**

The GTP 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 messages 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 result in more accurate answers [1]. Prompt-engineering will be discussed in the next section.

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.” [2].

Another parameter that can be optionally set is the *temperature*: this represents the degree of exploration or randomness of the model’s output. 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.

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']

**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 [3][4].

Prompt engineering has emerged as a powerful technique to enhance the performance and control the behavior of Language Models (LMs), particularly Large Language Models (LLMs), 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 [25].

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 [24][25][26], 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 application of *few-shot prompting* 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 involves 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 [24][26].

Appendix B shows the user messages used in the project.

1. Implementation

The following section 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.

* 1. PubMed scraping

The first step of the project consisted in collecting the necessary textual data for testing the causal discovery capabilities of the GPT LLM.

The used data is taken from the PubMed database, a free search engine accessing primarily the MEDLINE database of references and abstracts on life sciences and biomedical topics [16].

Only abstracts (and extra details) were extracted from the PubMed database.

**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 2.

The pipeline allows the user to extract textual data from PubMed by searching for specific terms.

Immagine che contiene testo, schermata, Carattere, design

Descrizione generata automaticamente

Figure 2: 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.

***search\_by\_terms***

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 allow up to 10 requests per second and 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 [17].

***get\_articles\_data***

The *get\_articles\_data* procedure is the second step of the data acquisition pipeline. It 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.

***clean\_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.

* 1. 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.

**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 flow is shown in Figure 3. 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.

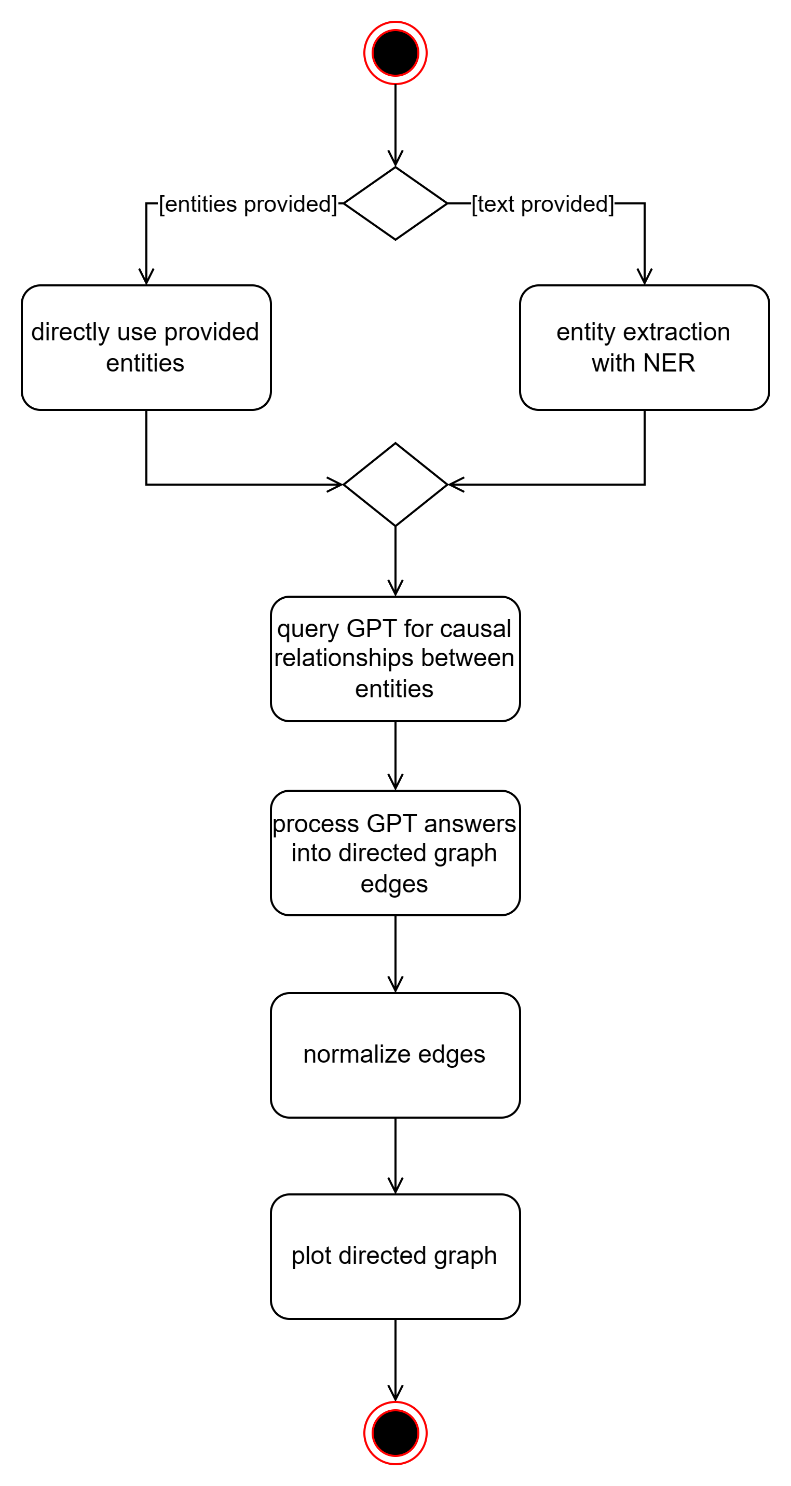


Figure 3: Causal discovery pipeline flow

**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.

**NER GPT prompt messages**

To enhance the performance of the Language Model for the NER 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 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 input prompt messages.

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.

**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 input text from the main pipeline 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 [18]. 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.

**Possible answers**

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:

1. "X" causes "Y";
2. "Y" causes "X";
3. "X" and "Y" are not causally related;
4. there is a common factor that is the cause for both "X" and "Y"

Appendix B.3 shows the full user message used for the causal discovery process.

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 [5][8].

The LLM is queried for each pair of variables.

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

in the case of ten entities, the total queries are .

**Double variables edge test**

The pipeline also allows for a double test for each variable pair, 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

With ten entities, the total queries are .

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 with the inconsistent answers obtained earlier, seeking the most likely relationship between the given variables. Appendix B.4 shows the complete user message used for this step. The newly acquired answers are then appended to the previously identified "valid" edges.

The output of this pipeline step is an array containing the type of causal relationship between each pair of entities.

**Performance considerations**

In terms of performance, this step was identified as the operational bottleneck within the entire causal discovery pipeline. 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.

* 1. Plotting the causal graph

The upcoming and final operations of the project involve preparing for the construction of the resulting causal graph. Appendix A.3 presents the source code of these functions.

**Graph preprocessing**

Before plotting the graph, the pipeline performs an intermediate operation of edge and node preprocessing. This step aids in the next ones by decoding the LLM answers and converting them into sets of nodes and normalized directed edges, represented in the form of “*X → 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 → 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.

**Cycle check**

The resulting graph should be a DAG (Directed Acyclic Graph), 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 a 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 Python package are primarily implemented in C++ to optimize performance [19].

The cycles are identified using the *all\_circuits* function from the *graph-tool* package.   
In case cycles exist, they are represented as lists of graph nodes and returned as an output parameter of the *find\_cycles* function.

**Plotting the graph**

The operation of 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 visualize 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 [20], while Pyvis serves as a visualization library suitable for generating interactive network graphs [7].

The function allows cycle highlighting and the creation of interactive plots. Cycles within the graph can be highlighted by coloring relevant edges in red.

The function supports both static and interactive modes of graph presentation, simplifying the visualization and analysis of entity relationships. The final interactive graph is then exported as an .html file.

Figure 4 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.

**Immagine che contiene linea, diagramma, cerchio, design

Descrizione generata automaticamente**

Figure 4: Causal graph example

* 1. 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.   
Appendix D shows all command line options.

1. Benchmarks and Metrics

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 [1].

* 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 [9]), as well as more specialized domains that require expertise in a specific field to ensure accurate understanding and interpretation (e.g., Neuropathic pain dataset [10]).  
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 [4]. Furthermore, using advanced Language Models, such as GPT-4, along with these prompt engineering techniques results even in higher accuracy.

* 1. 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 [1].  
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 [5][11] 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.

* 1. 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) [23] and more. Table # [12] lists the evaluation metrics used for the benchmark tests.

|  |  |
| --- | --- |
| **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. |
| Recall | Measures the ability to identify all actual causal relationships. |
| F1 score | Harmonic mean of precision and recall |
| 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. |

1. Results and Discussion

The following section analyzes the results of the benchmarks and real medical texts, discussing them and addressing the limitations found.

* 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 1 shows the results of the different methods and algorithms applied to the benchmark tests.

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 distance 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.

Appendix C.2 shows additional benchmark results, with the previously presented evaluation metrics.

Table 1: Benchmark results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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** |

* 1. 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 [18].

Appendix E shows a full example of the causal analysis process applied to a real-world medical text.

* 1. 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.

As previously stated, 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 [5]. This can introduce further bias and lead the LLM to produce inaccurate or misleading answers.

Furthermore, the complex nature of causal relationships, particularly in domains with intricate interdependencies like the medical field, may present challenges for LLMs to consistently capture the nuanced dynamics of causation.

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 a specific pair of variables.

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.

1. Conclusion

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 [21]. 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.

* 1. Future work

Since this project focused primarily on identifying causal graphs, future efforts could be focused on ensuring the acyclicity of 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 involves 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.

Appendix

1. Implementation detail

The source code used for this project is available at **repository\_url**.

* 1. Scraping

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

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

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)

* 1. Causal discovery

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

Complete *causal\_discovery\_pipeline*.

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.

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 function for GPT answers, when using double variable control with causal relationship between (‘A’ - ‘B’) and between (‘B’ - ‘A’).

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 = []

    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.

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.

def correct\_invalid\_edges(invalid\_edges, text=None, use\_pretrained\_knowledge=False):

    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

* 1. Plotting the causal graph

Decode GPT answers in directed edge.

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), given GPT answer.

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.

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 c in gt.all\_circuits(g):

        cycles.append([v\_prop[v] for v in c])

    return cycles

Build resulting causal graph.

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')

1. 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 [24].

* 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>[entity1]</Entity><Entity>[entity2]</Entity></Answer>

    (e.g., <Answer><Entity>diabetes</Entity><Entity>hypertension</Entity></Answer>)

    Text:

    <Text>{text}</Text>

* 1. 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.

* 1. 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' [8].

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>).

* 1. 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’* ”.

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>).

1. Benchmarks

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

* 1. Ground truth graphs

Immagine che contiene nero, oscurità

Descrizione generata automaticamente

Figure 5: Alcohol

Immagine che contiene nero, oscurità

Descrizione generata automaticamente

Figure 6: Cancer

Immagine che contiene nero, oscurità

Descrizione generata automaticamente

Figure 7: Asia

Immagine che contiene nero, oscurità

Descrizione generata automaticamente

Figure 8: Diabetes

Immagine che contiene nero, oscurità

Descrizione generata automaticamente

Figure 9: Obesity

Immagine che contiene nero, oscurità

Descrizione generata automaticamente

Figure 10: Smoking

* 1. Additional benchmark results

Table 2: Baseline results

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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** |

Table 3 GPT-3.5 results

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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** |

Table 4 GPT-4 results

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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** |

Immagine che contiene testo, schermata, Policromia, linea

Descrizione generata automaticamente

Figure 11: F1 scores for benchmarks

1. 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.

Examples:

  python causal\_llms.py ex                               # Example test.

  python causal\_llms.py b --algorithm {b|gpt}         # Benchmarks with algorithm.

  python causal\_llms.py s                                # Run the scraping process.

  python causal\_llms.py c --data-path </path/to/data>    # Causal analysis with data.

  python causal\_llms.py sc                               # Scraping & causal analysis.

The `algorithm` parameter specifies the algorithm to use for the benchmark tests.

The possible values are:

\* `b`: Baseline algorithm

\* `gpt`: GPT LLM

1. Example with real medical text

This appendix section shows a complete example of the causal discovery process applied to a real medical text [22] extracted from the PubMed repository.

**Article title:**

*Research progress on the protective mechanism of a novel soluble epoxide hydrolase inhibitor TPPU on ischemic stroke.*

**Original text:**

*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.*

**Extracted entities:**

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

**From the original text:**

*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.*

**Resulting causal graph:**

Immagine che contiene linea, diagramma, cerchio

Descrizione generata automaticamente

Figure 12: Causal graph real text example

Sources:

[1] [Causal Reasoning and Large Language Models: Opening a New Frontier for Causality (arxiv.org)](https://arxiv.org/abs/2305.00050), 02/06

[2] [GPT - OpenAI API](https://platform.openai.com/docs/guides/gpt), 29/06

[3] [Prompt Engineering Guide | Prompt Engineering Guide (promptingguide.ai)](https://www.promptingguide.ai/), 30/06

[4] [Investigating causal understanding in LLMs](https://www.lesswrong.com/posts/yZb5eFvDoaqB337X5/investigating-causal-understanding-in-llms), 02/06

[5] [Can large language models build causal graphs? (arxiv.org)](https://arxiv.org/abs/2303.05279), 12/06

[6] [[PDF] A Survey on Causal Discovery: Theory and Practice | Semantic Scholar](https://www.semanticscholar.org/reader/8733ea46dc4909017c7ad186d9c6202933c67b29), 01/06

[7] [Pyvis: Visualize Interactive Network Graphs in Python | by Khuyen Tran | Towards Data Science](https://towardsdatascience.com/pyvis-visualize-interactive-network-graphs-in-python-77e059791f01), 01/07

[8] [Causal Discovery with Language Models as Imperfect Experts (arxiv.org)](https://arxiv.org/abs/2307.02390), 14/07

[9] [Tubingen benchmark - Database with cause-effect pairs (mpg.de)](https://webdav.tuebingen.mpg.de/cause-effect/), 02/06

[10] [Causal-Discovery Performance of ChatGPT in the context of Neuropathic Pain Diagnosis](https://arxiv.org/pdf/2301.13819.pdf), 02/06

[11] [Bayesian Network Repository](https://www.bnlearn.com/bnrepository/), 20/07

[12] [Methods and tools for causal discovery and causal inference - Nogueira - 2022 - WIREs Data Mining and Knowledge Discovery - Wiley Online Library](https://wires.onlinelibrary.wiley.com/doi/full/10.1002/widm.1449), 01/06

[13] [On the Equivalence of Causal Models, Verma & Pearl](https://arxiv.org/ftp/arxiv/papers/1304/1304.1108.pdf), 02/06

[14] [Theoretical Impediments to Machine Learning, Pearl](https://arxiv.org/pdf/1801.04016.pdf), 02/06

[15] [Frontiers | Review of Causal Discovery Methods Based on Graphical Models (frontiersin.org)](https://www.frontiersin.org/articles/10.3389/fgene.2019.00524/full), 18/08

[16] [PubMed - Wikipedia](https://en.wikipedia.org/wiki/PubMed), 30/06

[17] [A General Introduction to the E-utilities - Entrez Programming Utilities Help - NCBI Bookshelf (nih.gov)](https://www.ncbi.nlm.nih.gov/books/NBK25497/), 02/06

[18] [Causal Parrots: Large Language Models May Talk Causality But Are Not Causal (matej-zecevic.de)](https://www.matej-zecevic.de/research/assets/Causal_Parrots__Large_Language_Models_May_Talk_Causality_But_Are_Not_Causal_.pdf), 05/06

[19] [graph-tool: Efficent network analysis with python (skewed.de)](https://graph-tool.skewed.de/), 10/08

[20] [NetworkX — NetworkX documentation](https://networkx.org/), 14/07

[21] [Understanding Causality with Large Language Models: Feasibility and Opportunities (arxiv.org)](https://arxiv.org/abs/2304.05524), 08/06

[22] [Research progress on the protective mechanism of a novel soluble epoxide hydrolase inhibitor TPPU on ischemic stroke - PubMed (nih.gov)](https://pubmed.ncbi.nlm.nih.gov/36846137/), 18/08

[23] [Causal Discovery Toolbox Documentation — Causal Discovery Toolbox 0.6.0 documentation (fentechsolutions.github.io)](https://fentechsolutions.github.io/CausalDiscoveryToolbox/html/index.html), 12/08

[24] [ChatGPT Prompt Engineering for Developers - DeepLearning.AI](https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/), 10/08

[25] [OpenAI and Andrew Ng Just Released a FREE ChatGPT Prompt Engineering Course | by The PyCoach | Artificial Corner](https://artificialcorner.com/openai-and-andrew-ng-just-released-a-free-chatgpt-prompt-engineering-course-b0884c03e946), 10/08

[26] [Best practices for prompt engineering with OpenAI API | OpenAI Help Center](https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api), 10/08