LUISS

Degree Program in Data Science and Management

Course of Advanced Statistics

Scaling Causal Discovery with Multi-Agent Large Language Models

Prof. Marta Catalano

SUPERVISOR

Prof. Alessio Martino

CO-SUPERVISOR

Gian Lorenzo Marchioni
ID 788811

CANDIDATE

Academic Year 2024/2025

Contents

Chapter 1

Introduction

Modeling cause-effect relationships is a foundational aspect of science. Causal inference allows researchers to study real-life processes accurately, and predict the outcomes of hypothetical scenarios. This branch of statistics finds applications in any field where researchers or engineers aim to exclude spurious correlations, understand root causes of phenomena or analyze the consequences of decisions.

A central challenge in causal inference is that the causal relationships within a system are rarely known. Causal discovery methods aim to recover such relationships directly from data. However, this task is challenging due to fundamental limitations: said relationships are identifiable only under strong assumptions, one of which being causal sufficiency. Scalability also poses a challenge, since the performance of current algorithms decreases dramatically with an increasing number of variables.

An effective way to overcome these limitations is the integration of prior knowledge. Domain experts often have information about the underlying system, which can help to resolve ambiguities that data alone cannot address and provide a significant head-start to statistical algorithms. Traditionally, this has been done by manually adding constraints. While effective in small systems, such an approach is not scalable to high-dimensional settings where hundreds of variables may interact. In this context, automated methods for incorporating prior knowledge are of great importance.

Large Language Models (LLMs) have recently emerged as a promising tool in this direction. Trained on vast amounts of unstructured text, LLMs encode a considerable amount of factual and relational knowledge, including causal information. This makes them a natural candidate for assisting in the construction of causal models. Recent work has shown that LLMs can support causal discovery tasks by providing candidate relations, reasoning over textual sources, or guiding the design of exper-

iments. At the same time, their integration raises important challenges, such as factual reliability, hallucinations, and the need for human oversight.

Other promising research directions are multi-agent and human-in-the-loop frameworks. Multi-agent systems allow different specialized LLMs to cooperate, each one focusing on a task such as partitioning the variables, retrieving external information, or running statistical algorithms. A human-in-the-loop approach ensures that expert knowledge can also guide an automated process, improving reliability.

This thesis explores these directions by investigating how causal discovery over many variables can be enhanced through the integration of prior knowledge, Large Language Models, and multi-agent systems. The central idea is to adopt a *Divide-and-Conquer* strategy, where the original high-dimensional problem is broken into smaller and more manageable subproblems. Each subproblem is then analyzed using a combination of data-driven methods, external knowledge retrieval, and human feedback, before being merged into a global causal graph.

1.1 Contribution

This thesis introduces an original *Divide-and-Conquer* method for large-scale causal discovery, in which Large Language Models are employed as agents to assist the decomposition, analysis, and reconstruction of high-dimensional causal graphs.

1.2 Research questions

The central questions addressed are:

I. Can LLM agents, augmented with external knowledge retrieval and human interaction, successfully contribute to causal discovery tasks?

II. Which steps of the *Divide-and-Conquer* pipeline are best tackled with LLMs?

III. Can their use be reconciled with existing theoretical guarantees from the literature?

IV. Can data-driven approaches and LLMs be combined in a complementary way?

1.3 Thesis structure

This thesis is organized as follows. Chapter 2 introduces the necessary background on causality, causal discovery methods, and Large Language Models. Chapter 3

reviews related literature, with a particular focus on large-scale causal discovery approaches and the role of language models and agentic systems in scientific tasks. Chapter 4 describes the proposed framework, detailing its components and design. Chapter 5 reports the experimental evaluation, highlighting strengths and limitations across a range of benchmarks. Finally, chapter 6 summarizes the contributions of the thesis and discusses possible directions for future research.

Note: The code of this project is not yet publicly available, as a conference submission based on this work is currently under review. Moreover, the actual implementation is intellectual property of Robert Bosch GmbH. This master's thesis was conducted at the Bosch Research Center in Renningen, Germany, under the supervision of Nicholas Tagliapietra, Dr. Jürgen Lüttin and Dr. Lavdim Halilaj.

Chapter 2

Background

This chapter sets the theoretical groundwork for the thesis, starting with Pearl's theory of causality, which formalizes Structural Equation Models (SEMs), Structural Causal Models (SCMs), and their induced causal graphs. Marginal and conditional independence are then reviewed together with d-separation and m-separation, followed by the main families of conditional independence (CI) tests.

Building on these notions, the Causal Discovery (CD) problem is defined and the two major approaches (constraint-based and score-based) are introduced, along with their limits in relation to observational data, Markov equivalence, and causal insufficiency (leading to MAGs and PAGs). Next, the chapter covers evaluation metrics in CD along with common benchmarks and simulators.

This chapter also motivates the need for large-scale CD, and lays out related theory on *Divide-and-Conquer* methods.

For completeness, a brief overview of Large Language Models (LLMs) is also included. It covers embeddings, emerging abilities, retrieval-augmented generation, prompting, and agentic systems.

2.1 Causality

A modern theory of causality is attributed to Pearl [1, 2], who introduced a unified, computable, and formal representation of causal assumptions and effects. The "pearlian" framework is based on Structural Equation Models, used in earlier research in social sciences [3, 4].

Example 2.1.1. Suppose a phenomenon is described by a model, for instance a set

of equations that links two random variables X and Y:

$$\begin{aligned} y &= \beta x + u_Y \\ x &= u_X \end{aligned} \tag{2.1}$$

For example, X could stand for a disease variable and Y for a certain symptom. In this case x stands for the level (or severity) of the disease, y stands for the severity of the symptom, and u_Y (value of a random variable U_X) stands for all factors, other than the disease in question, that could possibly affect Y when X is held constant.

Equation 2.1 may be flipped to quantify how the symptom X influences the disease Y.

Equations alone therefore do not imply the inherent "direction" that natural phenomena have, and do not necessarily possess causal semantics. This concept of "direction" is formalized within Structural Equation Models:

Definition 2.1.1 (Structural Equation Model (SEM))**.** [1, 2, 5] A SEM expresses the directionality of the underlying process. A SEM is denoted as a 4-tuple:

- A set of "explained" variables $\mathbf{V}$, called *endogenous variables*, X and Y in the above example (eq. 2.1)
- A set of "unexplained" variables $\mathbf{U}$, called *exogenous variables*, U_X and U_Y in the above example (eq. 2.1)
- A joint probability distribution of exogenous variables $P = \prod_{U_i \in \mathbf{U}} P(u_i)$
- A set of functions $\mathbf{F}$ (called structural equations), such as equations 2.1. Each one representing an autonomous mechanism governing a variable.

Exogenous variables $\mathbf{U}$ may be "observed" or "unobserved", in the latter case they serve as "errors" or "disturbances". In both cases, they are not influenced by other variables in the SEM.

Definition 2.1.2 (Structural Causal Model (SCM))**.** [2] Given a SEM with a set of functions $\mathbf{F}$, if each variable has a distinct equation in which it appears on the left-hand side (being the dependent variable), then the model is called a Structural Causal Model. This means that each endogenous variable is uniquely described by an autonomous mechanism, or a "law".

Though some subtle differences are present, in the literature the terms SEM and SCM are often used interchangeably [5].

Definition 2.1.3 (Causal Graph)**.** [6, 5] The set of structural equations $\mathbf{F}$ in a SCM induces a set of dependencies between variables. Those can be depicted as a graphical model, which we call the *Causal Graph*, which unmistakenly represents the "direction" of relationships in the structural equations using directed edges. Each equation in $\mathbf{F}$ has form $x_i = f_i(\tilde{\mathbf{V}} \subseteq \{\mathbf{V} \setminus X_i\}, \tilde{\mathbf{U}} \subseteq \mathbf{U})$. Thus, for each function f_i we can determine a set of "parents" for the variable X_i: $Pa(X_i) = \tilde{\mathbf{V}} \cup \tilde{\mathbf{U}}$. Doing so induces a graph representation of the SCM where we add a directed edge between each variable in $Pa(X_i)$ and X_I. This method of constructing it is also referred to as "Causal Edge Assumption": the value assigned to each variable X_i is completely determined by the function f_i given its parents.

For sake of simplicity, we will assume that the underlying causal graph will not contain cycles i.e. it is a *Directed Acyclic Graph* (DAG).

Conditional Independence

A key concept in Causality is that of conditional independence, which is linked to that of conditional probability. Given two random variables X, Y, we denote with $P(X)$ and $P(Y)$ their marginal probability distributions, and with $P(X|Y)$ and $P(Y|X)$ their conditional probability distributions of one given the other and vice-versa. These are of course linked by Bayes's theorem, where marginal distributions represent our prior knowledge about X and Y, while the conditional distributions represent more "informed" distributions. If X, Y are related or connected in some way, we expect that $P(X) \neq P(X|Y)$. However, this equality may occur in some situations.

Definition 2.1.4 (Marginal Independence [7])**.** Consider two random variables X, Y. An event x (drawn from $P(X)$) is independent of an event y (drawn from $P(Y)$) if $P(X = x|Y = y) = P(X = x)$. If this is true for all events drawn from $P(X)$ and $P(Y)$, we denote $X \perp\!\!\!\perp Y$ as "X marginally independent of X". Marginal independence is a symmetric property: if $X \perp\!\!\!\perp Y$, then $Y \perp\!\!\!\perp X$. The term "marginal" or "marginally" is commonly omitted when referring to this property.

A more common situation is when two events are independent given an additional event. Consider now a third random variable Z.

Definition 2.1.5 (Conditional Independence (CI))**.** [7] Consider three random variables X, Y, Z. An event x (drawn from $P(X)$) is conditionally independent of an event y (drawn from $P(Y)$), given a third event z (drawn from $P(Z)$ if $P(X =$

$x|Y = y, Z = z) = P(X = x|Z = z)$. If this is true for all events drawn from $P(X)$, $P(Y)$ and $P(Z)$, we denote $X \perp\!\!\!\perp Y|Z$ as "X conditionally independent of Y, given Z" or alternatively "X independent of Y, conditioned on Z".

Conditional independence is symmetric: if $X \perp\!\!\!\perp Y|Z$, then $Y \perp\!\!\!\perp X|Z$. More generally, instead of conditioning on a single r.v. Z, we could also use a set of multiple random variables $\mathbf{Z}$, called *conditioning set*. Conditional independence is weaker than marginal independence, the latter being the case where $\mathbf{Z} = \emptyset$.

Testing for CI between a pair of variables, given a conditioning set, is an important task in causal inference. As will be discussed in section 2.1.1, a whole family of algorithms relies on CI tests to find a causal graph. The most widespread types of CI tests are:

- Fisher's Z test [8]
- χ^2 test: Tests the null hypothesis that X is independent from Y given $\mathbf{Z}$. This is done by comparing the observed frequencies with the expected frequencies if X, Y were conditionally independent, using a χ^2 deviance statistic. The expected frequencies given independence are $P(X, Y, \mathbf{Z}) = P(X|\mathbf{Z})P(Y|\mathbf{Z})P(\mathbf{Z})$. The latter term can be computed as $P(X, \mathbf{Z})P(Y, \mathbf{Z})/P(\mathbf{Z})$ [9]
- G^2 test [10]
- Kernel-based tests: KCI [11] and FastKCI [12]
- Randomized Conditional Independence Test (RCIT) [13]

Some CI tests are catered to specific variable types: the χ^2 and G^2 tests apply to discrete variables, while Fisher's Z test assumes continuous Gaussian data. Kernel-based methods such as KCI and FastKCI, as well as RCIT, are non-parametric and can handle mixed or non-linear relationships. The choice of test depends on both variable type and computational efficiency.

Now, a useful definition that will later be relevant:

Definition 2.1.6 (Order of a CI test)**.** When performing a CI test between two variables, say X and Y, the *order* of a CI test refers to the cardinality of the conditioning set $|\mathbf{Z}|$. A CI test of order zero is therefore just testing for marginal independence ($\mathbf{Z} = \emptyset$).

Conditional independence allows for a convenient factorization of joint probability distributions of variables, leveraging Bayes' theorem and the chain rule for computing joint distributions [7]. If $X \perp\!\!\!\perp Y|Z$, it can be shown that $P(X, Y, Z) =$

$P(Z)P(Y|Z)P(X|Z)$. It is common to represent these relations using a DAG, where Z is a parent of both X and Y. This is the minimal example of a *Bayesian Network*, in particular a *naive Bayes* model, which in turn is a subtype of a larger family of *Probabilistic Graphical Models*.

The way CI is (sometimes ambiguously) represented in DAGs will be further discussed in the following chapter 2.1.1.

Bayesian Networks are related to SCMs in that both represent "direction" or "parent-child" relationships among random variables using a DAG. The first encodes joint probability distributions, while the latter encodes algebraic relationships of causal nature. In a sense, Bayesian Networks are simpler than SCMs, given that the objective is to conveniently compute probabilities, without implying the existence or attempting to estimate an exact quantitative "causal law". Bayesian Networks may nonetheless, under certain conditions, have a causal interpretation [2]. Within this thesis, this subtle distinction among the two families of models is neglected, assuming Bayesian networks are modeling causal phenomena.

2.1.1 The Causal Discovery Problem

In the previous sections 2.1 and 2.1, we introduced DAGs as a representation associated to SCMs or Bayesian Networks. However, the problem at hand is often to first determine the DAG, given a set of variables. As Pearl in [2] puts it, the DAG serves as a "blueprint" for later estimating the exact functions in a SCM. This task is referred to as *Causal Discovery* (CD) in the context of SCMs, or *Structure Learning* in the context of Bayesian Networks. In general, it is the problem of selecting a causal graph as a possible explanation for a given data set [5].

CD from datasets may be performed using a couple families of algorithms:

- Constraint-based methods: these methods usually start from a fully connected graph and prune edges after performing CI tests (described in section 2.1) on variable pairs. Examples are the Peter-Clark (PC)[14] and Fast Causal Inference (FCI)[15] algorithms.
- Score-based methods: these methods either search for individual graph structures or fit a model to the data, optimizing some evaluation metric. One example is DAGMA [16], whose basic idea is to fit either a Linear or a Multi-Layer Perceptron to estimate a structural equation for each variable in the dataset. Another is GES [17].

In the context of Causal Inference, data comes in three types:

- Observational Data: each sample is randomly generated from the (yet unknown) joint distribution of the variables. This is the most common scenario, in which data is passively collected
- Interventional (or experimental) Data: samples are collected while systematically setting a fixed value for one or more variables. Such an operation is called an *intervention*. This scenario corresponds to performing controlled experiments
- Counterfactual Data: hypothetical or "potential outcome" data describing what would have happened to the same units under alternative conditions (e.g. if a different treatment had been applied). These outcomes are not observed, but inferred via models

Since variables in a causal graph might not always be all measurable, another key concept in Causal Discovery is *Causal Sufficiency*.

Definition 2.1.7 (Causally Sufficient Variable Set)**.** [5] A set of variables $\mathbf{V}$ is said to be *causally sufficient* if and only if every cause of any subset of $\mathbf{V}$ is contained in $\mathbf{V}$ itself. This means there are no unobserved variables $\mathbf{U}$ that affect the behavior of the system generating the data set.

Unobserved variables might be *confounders*, meaning they are a common cause of two variables, or *mediators*, meaning they are an intermediate effect between two variables.
Causal Sufficiency is a rare condition in real datasets, and it inherently limits the CD process. If the observed process has hidden confounders, constraint-based methods will only be able to yield a *Mixed graph*, meaning some edges might turn out to be undirected ($X - Y$), bi-directed ($X \longleftrightarrow Y$) or have undefined endpoints ($X \circ\!\!-\!\!\circ Y$). If two nodes have a bi-directed edge between them, they are *spouses* of each other.

Both the availability of only observational data and causal insufficiency represent key obstacles to faithful causal discovery. It is worth addressing separately the theoretical limitations that each one introduces.

Causal Discovery on Observational Data

Any CD algorithm that uses observational data alone cannot uniquely determine a causal graph [18], but only an equivalence class of graphs called *Markov Equivalence Class*.

Before introducing it, some definitions are in order:

Definition 2.1.8 (Collider [5])**.** In a DAG, a node X is a *collider* if two arrow endpoints both point to it: $Y \longrightarrow X \longleftarrow Z$. X is a *shielded collider* if there is also a directed edge between Y and Z, otherwise it is an *unshielded collider* (also called a *v-structure*).

Definition 2.1.9 (d-separation [1])**.** Consider a subset of nodes $\mathbf{S}$ within a DAG. $\mathbf{S}$ is said to *block* a path π if either:

I. π contains at least one arrow-emitting (i.e. a "non-collider") node that is in $\mathbf{S}$

II. π contains at least one collider node that is not in $\mathbf{S}$ and has no descendants in $\mathbf{S}$

If a set $\mathbf{S}$ *blocks* all paths between two nodes X and Y, then $\mathbf{S}$ *d-separates* X and Y. If the DAG represents an SCM, it then holds that $X \perp\!\!\!\perp Y|\mathbf{S}$. If $\mathbf{S} = \emptyset$, then $X \perp\!\!\!\perp Y$.

Identifying v-structures is a key point in the *edge orientation rules* of constraint-based algorithms. After finding an undirected skeleton $\overline{G}$ through CI tests, suppose the d-separation sets for each non-adjacent pair of variables have been recorded in SepSet. The PC and FCI algorithms then apply the following orientation rules [14, 19, 20]:

I. For each triple of nodes X, Y, Z such that X and Y are adjacent, Y and Z are adjacent, but X and Z are not adjacent in $\overline{G}$, orient $X—Y—Z$ as $X \longrightarrow Y \longleftarrow Z$ if and only if $Y \notin \text{SepSet}(X, Z)$.

II. If $X \longrightarrow Y$, Y and Z are adjacent, X and Z are not adjacent, and the edge $Y—Z$ has no arrowhead at Y, then orient $Y \longrightarrow Z$.

III. If there is a directed path from X to Y and an undirected edge $X—Y$, orient it as $X \longrightarrow Y$.

IV. If $X \longrightarrow Y \longleftarrow Z$, $X—W—Z$, X and Z are not adjacent, and $W—Y$, then orient $W—Y$ as $W \longrightarrow Y$. This was later added by [21].

The first rule is applied once. Rules II-IV are applied repeatedly until no more edges can be oriented [20].

d-separation then links the structure of a DAG to a set of CI relations, also called an *independence model* [22]. Problem is that multiple DAG structures may imply the same independence model.

Definition 2.1.10 (Markov Equivalence Class (MEC) [23, 5])**.** Two DAGs G_1 and G_2 are *Markov equivalent* if and only if G_1 and G_2 have the same undirected "skeleton" and the same v-structures. This implies that they any pair of nodes that is d-separated (by any set of nodes) in G_1 is also d-separated (by any set of nodes) in G_2, and vice-versa. In some sources this property is called *observational Markov equivalence*.

Therefore, testing for CI alone does not allow to distinguish between two Markov equivalent graphs. Only through interventions it might be possible to solve ambiguous edge orientation. When only observational data is available, the best we can do is just to identify a MEC, as opposed to a unique DAG.

Since MECs are defined in terms of skeletons and v-structures only, edges that are not part of any v-structure remain undirected, resulting in a *Partially Directed Acyclic Graph* (PDAG).

This ambiguity in edge direction is also present in score-based CD algorithms, as common scoring criteria, such as BIC, assign the same score for multiple DAGs within a MEC [24]. Algorithms like GES (Greedy Equivalence Search) do not search over individual DAG but rather over equivalence classes, and thus output a PDAG [24].

Causal Discovery on Causally Insufficient Data

As noted earlier, if there are hidden confounders, the result of a constraint-based CD algorithm can at best be a mixed graph. If the data is also observational, it is necessary to extend the definition of MEC for this, more general, case.

Definition 2.1.11 (Ancestral Graph)**.** [22, 5] A mixed graph is *ancestral* if:

- There are no directed cycles. A directed cycle is simply a walk through the graph that returns to the original node, following directed edges,
- Whenever there is a bi-directed edge $X \longleftrightarrow Y$, then there is no other directed path from X to Y, or from Y to X,
- Whenever there is an undirected edge $X \longleftrightarrow Y$, neither X nor Y have any spouses or parents.

Ancestral graphs generalize DAGs by allowing bi-directed and undirected edges, making them suitable for representing systems with latent variables or hidden confounding.

Definition 2.1.12 (Maximal Ancestral Graph (MAG))**.** [22] An ancestral graph is *maximal* if, for every pair of non-adjacent nodes X, Y there exists a set of nodes $\mathbf{S}$ ($X, Y \notin \mathbf{S}$) that m-separates X and Y.

MAGs are "maximal" in the sense that no additional edges can be added without altering the set of conditional independence relations between variables. They are a useful causal structure as they explicitly acknowledge the presence of hidden confounders or other unobserved variables [22].

One may also extend the notions of collider and d-separation to MAGs:

Definition 2.1.13 (Collider (mixed graphs))**.** [5, 22] In a mixed graph, a node X is a *collider* if two arrow endpoints both point to it: $Y \longrightarrow, X \longleftarrow, Z$, $Y \longleftrightarrow, X \longleftrightarrow, Z$, $Y \longleftrightarrow, X \longleftarrow, Z$, or $Y \longrightarrow, X \longleftrightarrow, Z$.

Definition 2.1.14 (m-separation)**.** [5, 22] *m-separation* for mixed graphs is defined similarly to d-separation (see definition 2.1.9), but with colliders defined as in 2.1.13.

Finally, to represent equivalence classes of MAGs when only observational data are available, a different notation is introduced:

Definition 2.1.15 (Partial Ancestral Graph (PAG))**.** [22, 5] A PAG is a mixed graph that represents a Markov equivalence class of MAGs. It may contain directed, bi-directed, undirected, and circle endpoints ($\circ\!\!-, ...$) to denote uncertainty about edge orientation while preserving the encoded m-separation relations.

Like DAGs, multiple MAGs may actually imply the same conditional independence model. If we are also dealing with purely observational data: constraint-based CD algorithms output an all possible "valid" MAGs. More formally:

Definition 2.1.16 (Markov Equivalence Class (MEC))**.** [22, 23] In the more general case, if G_1 and G_2 are two MAGs, they are *Markov equivalent* if for all triplets X, Y and $\mathbf{S}$, where X and Y are nodes and $\mathbf{S}$ is a (potentially empty) set such that $X, Y \notin \mathbf{S}$, then X and Y are m-separated in G_1 if and only if they are m-separated in G_2.

As in with DAGs, the above condition implies that the two graphs encode the same set of conditional independencies. The difference between DAGs and MAGs is that having the same adjacencies and unshielded colliders, though necessary, is not sufficient for Markov equivalence of MAGs. Every non-maximal ancestral graph can uniquely be transformed to a Markov equivalent MAG, by appropriately adding bi-directed edges [22].

Similarly to how partially directed graphs use undirected edges to represent the "ambiguity" of edge orientation within a MEC for DAGs, a *Partial Ancestral Graph* (PAG) is used to represent a MEC for MAGs. A PAG is a mixed graph that also allows for undefined endpoints: ∘—... , ...—∘. Undefined endpoints express uncertainty when determining endpoints that do not affect the presence of a v-structure, and thus do not affect m-separation.

Lastly, a definition which will be useful in section 3.1.

Definition 2.1.17 (Residual)**.** Given an independent variable x, its *residual* when projected on another variable y is $y - E(y|x)$

Residuals are useful for identifying edge directions by testing whether a variable is independent of the residuals when projected onto its neighbors.

2.1.2 Evaluation in Causal Discovery

Evaluation of a CD algorithm is tricky. Even excluding the intrinsic theoretical limitations introduced in sections 2.1.1 and 2.1.1, in most real-life scenarios the exact DAG (let alone the full SCM) modeling the underlying data generating process is unknown. Within this thesis, it is assumed that a ground truth, consisting of an exact DAG, is available. As will be discussed in 2.1.2, this issue is a topic of research in and of itself.

Metrics

Given a set of variables $\mathbf{V}$ in a dataset, which corresponds to a set of nodes in the learned DAG or PAG, CD can be viewed as a binary classification task. In particular, we can define an "arrow precision" and "arrow recall" as follows [5]:
For each ordered pair (X, Y), $X, Y \in \mathbf{V}$, we check whether the oriented edge $\pi = X \longrightarrow Y$ exists in the ground truth G and in the learned graph $\tilde{G}$:

I. If $\pi \in G$ and $\pi \in \tilde{G}$, it is counted as true positive (TP)

II. If $\pi \notin G$ and $\pi \notin G$, it is a true negative (TN)

III. If $\pi \notin G$ but $\pi \in \tilde{G}$, it is a false positive (FP)

IV. If $\pi \in G$ but $\pi \notin \tilde{G}$, it is a false negative (FN)

This method basically compares "arrow" endpoints for all edges in both G and $\tilde{G}$, and is used also when evaluating a PAG: "undefined" endpoints (∘—... , ...—∘) are

not considered valid "arrow" endpoints as to not overestimate algorithm performance [5].

After counting TP, TN, FP and FN values, we can calculate the *ArrowHead Precision* (AHP) and *ArrowHead Recall* (AHR) as the ratio between correctly predicted arrowheads over total predicted arrowheads, and ratio of correctly predicted arrowheads over true arrowheads respectively [5]:

$$\text{AHP} := \frac{TP}{TP + FP} \qquad \text{AHR} := \frac{TP}{TP + FN} \tag{2.2}$$

Another widespread metric in binary classification is the F1-score:

$$\text{F}_1 := 2 \cdot \frac{\text{AHP} \cdot \text{AHR}}{\text{AHP} + \text{AHR}} \tag{2.3}$$

It is also useful to define a sort of "edit distance" between G and $\tilde{G}$, called *Structural Hamming Distance* (SHD). It basically counts the number of necessary operations (addition, deletion or reversal of an edge) to transform G in $\tilde{G}$ [5]. Let $\bar{\mathbf{E}}(G - \tilde{G})$ be the difference between the edge sets of G and $\tilde{G}$, then the SHD is defined like so [5]:

$$\text{SHD}(G, \tilde{G}) := \sum_{(X,Y),\, X<Y}^{\mathbf{V}^2} \begin{cases} 1 & X \longrightarrow Y \in \bar{\mathbf{E}}(G - \tilde{G}), \\ 1 & Y \longrightarrow X \in \bar{\mathbf{E}}(\tilde{G} - G), \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Interpreting SHD requires some care, as the score itself is not normalized and lacks context. Suppose the ground truth graph G has n nodes and m edges. Suppose there are two "dummy" algorithms $\mathcal{A}$ and $\mathcal{B}$: $\mathcal{A}$ always outputs a fully connected graph (or clique) $\tilde{G}_{\mathcal{A}}$; $\mathcal{B}$ always outputs a graph with no edges at all $\tilde{G}_{\mathcal{B}}$. The SHD for these "trivial" cases is $\text{SHD}(G, \tilde{G}_{\mathcal{A}}) = n(n-1)/2 - m$ and $\text{SHD}(G, \tilde{G}_{\mathcal{B}}) = m$.

Benchmarks

As mentioned, performing a thorough evaluation of CD methods is challenging due to the scarcity of benchmarks with available ground truth.

The most used benchmarks in the literature are gathered in the `bnlearn` repository [1]. Most of them belong to the "discrete bayesian networks" group, meaning that each node in the causal graph represents a discrete (or categorical) variable. In "gaussian" networks, all variables are continuous and all of the conditional probability distri-

[1] BN Repository

butions are Gaussians. "Conditional linear gaussian networks" include both discrete and continuous variables, with the restriction that continuous variables cannot be parents of a discrete one [25].

Another common benchmark is the *Neuropathic Pain* simulated dataset [26] [2], which also falls in the "discrete" category. It is an especially useful benchmark given its size (222 nodes, 770 edges).

Given a graph, independent observations (i.e. data) can be simulated in a process called *sampling*, the simplest being *forward* (or *ancestral*) *sampling*[27]. The graph is treated as a causally sufficient model, with predefined conditional probabilities for each node. The process traverses the graph in topological order, starting with root nodes (with no parents), and samples values for each variable based on its conditional probability distribution and the already-sampled values of its parents [7, 28]. Forward sampling is simple when variables are discrete, but other approaches like *importance sampling* are needed when generating continuous variables [7].

Another way of benchmarking is to generate random discrete or gaussian DAGs using the Erdős–Rényi model [29] or Barabási-Albert model [30].

With both aforementioned methods, however, the generated observations are intrinsically causally sufficient: they are sampled from a "sufficient" model where all variables are endogenous or, at least, observed.

The size of a causal graph, meaning the amount of nodes $n = |V|$, is also a key pain point in benchmarking CD methods, and the one that this study attempts to address. Both constraint-based and score-based methods scale poorly, albeit for different reasons:

- Constraint-based methods rely on CI test. Consider for instance FCI [15]. It starts from a fully connected undirected graph and for each pair of variables $X, Y \in \mathbf{V}$ it tries to find a conditioning set $\mathbf{Z} \subset \mathbf{V} \setminus \{X, Y\}$ to try and remove the edge X—Y if a CI test finds $X \perp\!\!\!\perp Y|\mathbf{Z}$. A back-of-the-envelope calculation reveals that this single search alone requires $(n-2)!$ CI tests. Even though removing edges does restrict the search space of $\mathbf{Z}$ to just nodes still adjacent to X, this type of strategy does not scale to large values of n.
- Score-based methods try to directly fit a model to data and optimize a score such as the BIC. The number of possible network structures grows exponentially with the number of variables and exhaustive search for the optimal struc-

[2] Available on GitHub

ture becomes prohibitively costly [31].

Dealing with large scale CD tasks is possible, for instance by leveraging domain-specific knowledge, including chronological order of variables, to rule out causal connections [32]. In general, restricting the search [3] in some way is the common denominator for increasing efficiency. "Greedy" score-based methods like GES [17] also have an accuracy problem, as they may not converge to a global optimum.

The CausalMan Benchmark

The main benchmark used within this study is *CausalMan* [33], a novel data simulator designed to evaluate CD and other causal inference tasks. CausalMan generates realistic data in the context of a manufacturing line, and improves on existing benchmarks from a few angles:

- **Hybrid-datatypes:** Data are of mixed type (discrete and continuous) with no restrictions
- **Causal Insufficiency:** It features many unobserved exogenous variables, with complex relationships between each other and with endogenous variables. The generated data is thus causally insufficient
- **Non-linearities:** The equations in its SCM are accurate from a physics standpoint, often being non-linear
- **Large-Size:** Size is considerable, with the largest dataset (*CausalMan Medium*) having $n = 186$ nodes and $m = 381$ edges in the observable part of the ground truth

2.1.3 Large Scale Causal Discovery - Divide&Conquer

In the previous section 2.1.2, the issue of CD on observational datasets with a high number of variables was introduced. It appears clear, both from literature [34, 35, 30] and experiments performed in this study (see section 5), that many algorithms either fail to terminate in reasonable time or yield poor performance in terms of F_1 and SHD scores.

Divide-and-conquer approaches are a solution often explored in literature; the method proposed in this study also falls within this category. It is thus appropriate to provide an introduction to existing definitions and terminology.

[3] Searching for edges in score-based methods or conditioning sets in constraint-based methods

As the name suggests, the basic principle is to divide the task into smaller and easier sub-tasks, and finally combining all the partial results together into a final solution. In particular, these methods can be characterized by three distinct steps:

1. **Divide**: Partition the set of variables into smaller groups, often recursively
2. **Conquer**: Run a causal discovery algorithm to find a "local" causal graph (either a DAG, PAG, or MAG) for each group
3. **Merge** the results together, solving conflicts and/or adding edges to connect the "local" causal graphs

The whole approach hinges on the quality of the subdivision of the problem (i.e. the *Divide* step) and later on how well the individual sub-tasks are solved (*Conquer* step). What properties should a partition have, such that subdividing the variables set will not result in a loss of information?

Among others, one approach to partitioning has been proposed by [36]. In their study, [36] also provide a formal definition.

Definition 2.1.18 (Causal Partitioning ([36]))**.** Let $\mathbf{V} = \{u, v, \dots\}$ be a variable set that is also the set of nodes in a DAG $G(\mathbf{V}, \mathbf{E})$. Consider a set of variable subsets $\mathcal{P} = \{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \dots\}$, with $\mathbf{P}_i \subset \mathbf{V}$.
$\mathcal{P}$ is a *causal partitioning* of $\mathbf{V}$ if and only if all the following are true:

I. $\bigcup_i \mathbf{P}_i = \mathbf{V}$,

II. $\forall\, u, v \in \mathbf{P}$: if $\exists\, \mathbf{P}_i, \mathbf{P}_j \subset \mathbf{S}$ such that $u \in \mathbf{P}_i$, $v \notin \mathbf{P}_i$, $u \notin \mathbf{P}_j$, $v \in \mathbf{P}_j$; then u and v are not adjacent in G,

III. $\forall\, u, v \in \mathbf{V}$: if u and v are not adjacent in G; then either:

   (a) $\exists\, \mathbf{P}_i, \mathbf{P}_j \subset \mathbf{S}$ such that $u \in \mathbf{P}_i$, $v \notin \mathbf{P}_i$, $u \notin \mathbf{P}_j$, $v \in \mathbf{P}_j$,

   (b) $\exists\, \mathbf{P}_k \subset \mathbf{S}$ such that $u \perp\!\!\!\perp v \,|\, Z$, where Z is a conditioning set $Z \subset \mathbf{P}_k$.

To give a more "informal" explanation, the three conditions above mean that:

I. Each variable in $\mathbf{V}$ should be assigned to at least one group.

II. If two variables are adjacent in G (i.e. they are connected by an edge), then they must either appear together in a group, otherwise only one of the two is "allowed" to appear alone in other groups. In other words, there must not be a pair of groups that divides a pair of adjacent variables.

III. If two variables are not adjacent in G, then they should be assigned to different groups at least once, or they must be d-separated in at least one group.

Note that the conditions in definition 2.1.18 allow for groups $\mathbf{P}_i$ to overlap. Actually, unless G has multiple connected components, the only way to obtain smaller groups that comply with condition 2 is to have some overlapping nodes.

This fact is quite relevant, as it dictates the operations to be performed in the merge phase: with disjointed groups, the merging phase should introduce connecting edges. On the contrary, if there are substantial overlaps between groups, then the merge phase is about solving mismatches between the smaller graphs, as in [36].

Another work by [34] provides an alternative definition of how an overlapping graph partitioning should look like:

Definition 2.1.19 (Causal Cut [34])**.** Consider a variable set $\mathbf{V}$, and three subsets of it: $\mathcal{P} = \{\mathbf{C}, \mathbf{P}_1, \mathbf{P}_2\}$, with $\mathbf{C}, \mathbf{P}_1, \mathbf{P}_2 \subseteq \mathbf{V}$. $\mathcal{P}$ is a valid *causal cut* over $\mathbf{V}$, if and only if both of the following are true:

I. $\mathbf{P}_1 \cup \mathbf{P}_2 \cup \mathbf{C} = \mathbf{V}$

II. $\forall u \in \mathbf{P}_1$ and $\forall v \in \mathbf{P}_2$, there exists a variable set $\mathbf{C}_{uv} \subseteq \mathbf{C}$ such that $u \perp\!\!\!\perp v | \mathbf{C}_{u,v}$. This means that variables in $\mathbf{C}$ d-separate every variable in $\mathbf{P}_1$ from the ones in $\mathbf{P}_2$

Given that a suitable *causal cut* exists, the groups that $\mathbf{V}$ is split into would be $\mathbf{P}_1 \cup \mathbf{C}$ and $\mathbf{P}_2 \cup \mathbf{C}$, meaning the groups overlap.

Lastly, a more "practical" definition of Causal Partitioning is given by [37]:

Definition 2.1.20 (Causal Partitioning)**.** [37] Given a DAG $G = (\mathbf{V}, \mathbf{E})$, the aim to partition the node set $\mathbf{V}$ into two (overlapping) sub-groups $\mathbf{P}_1$ and $\mathbf{P}_2$ with the following constraints:

I. $\mathbf{P}_1 \cup \mathbf{P}_2 = \mathbf{V}$

II. $\forall v \in \mathbf{P}_1 \setminus \mathbf{P}_2$ and $\forall u \in \mathbf{P}_2 \setminus \mathbf{P}_1$, it holds that $(v \to u) \notin E$ and $(u \to v) \notin \mathbf{E}$

In other words, there should not be any edges between variables in different groups, barring their intersection.

The "practical" definition 2.1.20 also fits the method and the visualizations in [30], which actually provides another definition given a starting undirected superstructure, however it is less useful in this study.

As briefly stated, partitioning may be done recursively. For instance, in the first step the full variable set $\mathbf{V}$ might be divided into $\mathcal{P} = \{\mathbf{P}_1, \mathbf{P}_2\}$. Subsequently, $\mathbf{P}_1$ can itself be partitioned, yielding $\mathcal{P}' = \{\mathbf{P}_{1.1}, \mathbf{P}_{1.2}, \mathbf{P}_2\}$. Once the process terminates, the sequence of partitions can be represented as a *partition tree*, where each node corresponds to a partition. The smallest, terminating partitions in $\mathcal{P}$ will be called *leaf partitions*, while partitions derived from the same parent set (whether $\mathbf{V}$ or some $\mathbf{P}_i$) are called *sibling partitions*. In the example, $\mathbf{P}_{1.1}, \mathbf{P}_{1.2}, \mathbf{P}_2$ are all leaf partitions; $\mathbf{P}_{1.1}$ and $\mathbf{P}_{1.2}$ are siblings, and so are $\mathbf{P}_1$ and $\mathbf{P}_2$.

Having covered three sets of criteria (definitions 2.1.18, 2.1.19 and 2.1.20) on how a set of causally connected variables should be partitioned, to benchmark different methods it is useful to define some quantitative metrics.

The first one is the *Causal Cut Error*:

Definition 2.1.21 (Causal Cut Error (CCE) [34])**.** Consider a DAG $G = (\mathbf{V}, \mathbf{E})$ and a partitioning $\mathcal{P} = \{P_1, P_2, ...\}$, with $P_i \subset \mathbf{V}$. Let $N = |E|$ be the number of pairs of variables that are connected by an edge in G. Let N_e be the number of pairs of variables that are connected by an edge in G, but do not appear together in any $P_i \in \mathcal{P}$. More formally, N_e is computed as follows:

$$
\begin{aligned}
&N_e \leftarrow 0 \\
&\text{for each ordered pair } (X, Y) \in \mathbf{V} : \\
&\quad \text{if } X \longrightarrow Y \in \mathbf{E} \text{ then :} \\
&\quad\quad \text{if } \{X, Y\} \not\subseteq P_i \text{ for all } P_i \in \mathcal{P} \text{ then :} \\
&\quad\quad\quad N_e \leftarrow N_e + 1
\end{aligned}
$$

The *Causal Cut Error* is therefore:

$$\text{CCE} := \frac{N_e}{N}$$

Intuitively, the CCE is the ratio between the number of pairs of connected variables that have been (wrongly) split across different groups, over the total number of connected variable pairs. Unless the ground truth graph G has multiple connected components, obtaining a CCE close to zero requires overlapping causal partitioning. Otherwise, it would be expected that a disjoint partitioning "cuts" at least some valid edges.

Another metric to evaluate a partitioning is the overlap ratio:

Definition 2.1.22 (Overlap Ratio)**.** Given two partitions $\mathbf{P}_1, \mathbf{P}_2 \subset \mathbf{V}$, we can also define the *overlap ratio* as

$$\text{OR} := \frac{|\mathbf{P}_1 \cap \mathbf{P}_2|}{|\mathbf{P}_1 \cup \mathbf{P}_2|}$$

In section 5, both CCE and average overlap ratio between sibling partitions will be computed and used to compare different approaches for the *Divide* step.

2.2 An Overview on Large Language Models

The term Large Language Model (LLM) refers to a class of pre-trained text inference models based on the transformer architecture [38, 39]. "Large" refers to the amount of parameters (or *weights*), usually in the order of billions [39]. While a "language model" is not a new concept, LLMs have evolved beyond simply generating text, instead focusing on solving complex tasks [39].

A detailed explanation of their functioning, training and evaluation goes beyond the scope of this thesis. For what concerns this specific use case, the following sections will just provide a brief introduction to embedding models, emerging abilities of LLMs, Retrieval-Augmented-Generation, Multi-Agent Systems, and relevant prompting techniques.

2.2.1 Embedding models

In the context of language models, *embedding* refers to a series of techniques to represent text (be it a token, word, phrase or document), as a dense, high-dimensional vector of real numbers [38].

Many models, varying in efficacy and complexity, have been introduced over time to output text embeddings, examples include Word2Vec, BERT, with the most advanced being LLM-based embeddings [38, 40, 41]. The overall objective is for the vector representation to encode semantic similarity between pieces of text, with more sophisticated models being able to take into account additional context [38]. More nuanced text representations are desirable in many applications, including information retrieval [40] and clustering of text [42].

Embeddings are also a core component of LLMs: any text that they receive as input, or generate as output, exists as a sequence of tokens and their embeddings within the model [38].

2.2.2 Emerging abilities of LLMs

First, a couple "technical" definitions: the entire input+output token sequence of an LLM is usually referred to as "*context*". Due to the way LLM interfaces are often designed (either apps or APIs) the user may typically add his own text (called "*prompt*") to a pre-existing context, although this distinction is not well-established.

The key to the widespread application of LLMs stands in their ability to generalize beyond narrow natural language processing tasks. State-of-the-art LLMs excel at dialogue, answering complex questions, coding and making decisions or recommendations [43]. This can be linked to so-called *emerging abilities*, namely [39]:

- *In-Context Learning*: Assuming that the model is provided with a prompt containing detailed instructions and several examples, it can generate the expected (and good quality) output by exploiting patterns in the input, with no additional training [39, 44].
- *Instruction Following*: Combining supervised fine-tuning on a set of `(prompts, expected outcome)` pairs for a variety of different tasks and reinforcement learning with human feedback has been shown to improve the generalization of LLMs, even on tasks not appearing in fine-tuning data and without examples [39, 45, 46].
- *Step-by-step reasoning* (or just *reasoning*): Guided by appropriate fine-tuning and/or prompts, LLMs can solve more complex problems by predicting intermediate answers and/or logical steps in the output sequence, leading to the final result [39, 47]. This output behavior is referred to as "reasoning" or "thinking", and can be achieved with prompting alone (see section 2.2.4), in which case it falls within the *In-Context Learning* ability [48].

Building on top of these abilities, a fairly recent advancement introduced by [49] and popularized by OpenAI [50, 51], has been to tune LLMs to format their output is such a way that it can easily be parsed and used to pass arguments to other code. "Other code" could be an API or a user-defined function, and as such this behavior is termed *function* or *tool calling*. These *tools* can be regarded as a way to extend an LLM's knowledge or capability, albeit current models behave inconsistently, exhibiting both over- and under-reliance on tools [52].

Another property of LLMs, though subject of scrutiny [53, 54], is their *factuality*. The term denotes a model's capability of generating content grounded in reliable sources [54], acting as a knowledge base [55]. This could also be considered an

"emerging" property, stemming not from model size in terms of parameters, but rather from the size of its pre-training corpus. Corpora for general purpose models may include trillions of tokens [47] worth of text on various domains scraped from sources like Wikipedia, Reddit, miscellaneous web pages, news and textbooks [56].

Factuality is closely related to the issue of *hallucinations* in LLM outputs, which can be defined "baseless or unwarranted content" [53] or "content that appears nonsensical or unfaithful to the provided source" [54]. The cited studies actually distinguish between "factual/non-factual" and "non-hallucinated/hallucinated" output, as generated content may be factually true, but irrelevant to the prompt or it may be relevant but vague or outdated.

The key takeaway from the cited factuality surveys [53, 54] is that LLMs do internalize factual knowledge to a significant extent, as shown for instance by GPT-4 reaching 86.4% accuracy on the MMLU (a multi-task multiple choice test benchmark [57]) and passing the USMLE (United States Medical Licensing Examination) by over 20 points, or by LLMs attaining up to 71% factual precision (FActScore [58]) in biography generation. However, reliability is still an issue: on adversarial truthfulness benchmarks such as TruthfulQA [59], GPT-4 achieves only about 29% truthfulness. Factuality issues are attributed due to outdated or low-quality data in pre-training corpora and to "snowballing" hallucinations. In section 2.2.3, Retrieval Augmented Generation will be introduced as a possible (and widespread) solution [53].

2.2.3 Retrieval-Augmented Generation (RAG)

Introduced by [60] as the integration of external, "non-parametric" memory (also known as a *knowledge base*) accessed by a *retrieval* system, to complement the "parametric" memory of a pre-trained sequence-to-sequence transformer, i.e. a *generator*.

The retrieval system itself usually includes several components. Suppose the knowledge base is a set of documents: the goal is to provide the generator with the most relevant ones relative to the topic(s) in its input text. To achieve this, [60] treated the input sequence of the generator as a "query" to the document knowledge base. Thus, [60] implemented a retriever composed of two BERT models that embed documents and queries into the same space, and a ranking based on embedding similarity between query and documents.

Similar overall blueprints are followed my most recent RAG pipelines, though most

skip the fine-tuning performed in the introductory paper [60], as many LLM-based embedding models are designed and benchmarked for text retrieval [40, 41, 61].

More sophisticated retrieval approaches employ multiple LLMs to fetch information from sources and summarize a useful passage for the generator [62], an example will be discussed in section 3.3.

2.2.4 Prompting Techniques

In section 2.2.2 *prompts* and *in-context learning* were defined. These aspects alone are a field for experimentation and research. There is huge variety of what can be packaged as input to an LLM, considering that typical state-of-the-art models have a maximum *context length* (which is the amount of tokens of the input+output sequences, also called *context window*) in the order of 100k-200k tokens. Notably, LLM's in-context learning ability and general response quality tends to drop with increasing context lengths [63, 64].

Empirically, simple questions and tasks can be successfully solved with a straightforward prompt, that does not include examples: this is named *zero-shot* prompting. More complex tasks benefit from leveraging in-context learning through more elaborate prompts. The simplest approach is *few-shot* prompting, that is including additional context and one or more examples of `(prompt+expected output)` pairs [56, 65].

More sophisticated prompting techniques include *Chain-of-Though* (CoT) and *Reasoning+Acting* (ReAct) prompting. Both are a ways to condition an LLM to perform step-by-step reasoning through in-context learning. CoT is an extension of few-shot prompting, in which multiple examples of input and expected output sequences are provided. The output examples also include a series of simple, intermediate "thinking" steps in natural language, mimicking how a human would tackle problems [66, 56]. CoT has been shown to be a strict improvement over few-shot prompting on arithmetic, common sense and symbolic reasoning benchmarks [66]. ReAct is a further extension of CoT, where an LLM is conditioned to output verbal "reasoning" alternated with "acting", meaning interacting with an environment or "action space", and "observing" the result of actions [67]. In two experimental settings in the pioneering paper [67], the "action space" consists of calling the Wikipedia API to look up information or finalizing the answer. Quoting [67], the example output sequences include a combination of "thoughts" that:

- Decompose questions (“I need to search x, find y, then find z”)

- Extract information from Wikipedia observations (“x was started in 1844”, “The paragraph does not tell x”)
- Perform commonsense (“x is not y, so z must instead be...”) or arithmetic reasoning (“1844 < 1989”)
- Refine a search (“maybe I can search/look up x instead”)
- Synthesize the final answer (“...so the answer is x”)

"Acting" can be interpreted as a generalization of tool calling, introduced in section 2.2.2. In practical applications, the two concepts can be considered synonymous[4]. Thus, ReAct can pragmatically be treated as CoT prompting where output sequence examples include appropriate tool-calling, as shown in the example prompts in [67] and [68][5]. ReAct is also a way to introduce a sort of "active" RAG, contrasting hallucinations and factuality errors [67].

2.2.5 LLMs together strong - Agentic Systems

Following the introduction of tool-calling and advancements in inducing "reasoning" in LLM outputs (either through prompting or as an "inherent" feature through post-training), a recent concept in literature and industry media is that of "*LLM Agent*".

An informal definition is the following. An *agent* is a LLM (or any AI system) able to perform "reasoning", "planning" and "actions" autonomously [68, 62]. An agent is able to interact with tools, the user and often other agents, and has access to a memory of past interactions [62].

In *multi-agent*, or *agentic*, systems each LLM instance (i.e. each agent) has a unique role as a consequence of a different set of prompts and available tools. Their interaction may follow several design patterns, such as [62]:

- Chaining pattern: multiple agents, in sequence, perform a different task each, usually extending or editing the output of previous agents
- Router and Orchestrator-Workers patterns: a central agent creates and/or assigns tasks to specialized agents
- Evaluator-Optimizer pattern: one agent generates output, another evaluates it and provides feedback to the first

[4]See for instance the ReAct agent implementation in LangChain
[5]Prompts are also available on GitHub

Empirically, multi-agent systems yield better results compared to a single-LLM baselines, especially on tasks that benefit from agent specialization. Splitting a task across multiple agents is also a more optimal use of LLMs' finite context windows. [69] show significant accuracy improvements of their multi-agent "debate" system, without any tools, compared to a single model on the same zero-shot prompts. Tests were conducted on a variety of benchmarks, including the aforementioned MMLU [57], GSM8K (grade school math problems) [70], arithmetic, biographies and chess, implying a factuality improvement as well. Multi-agent applications built using Microsoft's AutoGen framework also exhibit improvements over single agents systems in its introductory paper [71]. Specifically, [71] reports higher success rates over single tool-equipped LLMs, some also using ReAct prompting, on math problems, the ALFWorld (decision making in household tasks) [72] and OptiGuide (coding and supply chain optimization) [73] benchmarks.

Chapter 3

Related Work

3.1 Large scale causal discovery

Section 2.1.3 introduced some definitions and evaluation criteria on divide and conquer approaches for CD, without describing a precise method. As mentioned, different algorithms may primarily be classified by the strategy used for each of the key phases in the process: the *Divide* step, the *Conquer* step and the *Merge* step. It is thus in order to provide a brief comparison of approaches discussed in literature, clearly delineating the strategy for each phase.

A first family of methods builds partitions through hierarchical clustering. [74] proposes a clustering strategy using a correlation-based distance metric, obtaining non-overlapping groups of variables. Local graphs for each partition are found through a constraint-based algorithm (PC [75]). Connections between groups are added through a two-phase strategy that first finds candidate edges using correlation between residuals and then filters them through CI tests. Candidates are are then selected and oriented using a modified BIC score criterion. Similarly, [76] applies hierarchical clustering, but treats discrete and continuous variables differently: in the former case, it employs a mutual information distance metric, while in the latter it too relies on correlation. During the clustering phase, some edges are also "blacklisted". Within each cluster, the PC algorithm [14] is applied to recover an undirected skeleton, excluding the blacklisted edges. Inter-group connections are later established using CI tests between pairs of nodes in different clusters. Final edge orientations are then obtained by finding v-structures. More recently, [77] proposed a modified hierarchical clustering based on correlation, merging together smaller clusters. PC [14] is then used in the conquer phase. Merging is carried out

similarly to [74].

A different class of methods relies on iterative CI testing to create a causal partitioning. [34] develops a partitioning algorithm based on finding conditioning sets that d-separate random pairs of variables. The conquer step can be carried out by any causal discovery algorithm that outputs a DAG; the authors use LiNGAM [78] in their experiments. The merge phase introduces a novel ranking scheme, where edges are ordered by significance scores (retrieved from the underlying CD algorithm). Conflicts are resolved by discarding edges inconsistent with higher-ranked ones, while redundant edges are checked by performing CI tests using existing paths as conditioning sets. [36] introduces CAPA, a recursive partitioning algorithm based on iterative higher-order CI tests (see definition 2.1.6). Once groups are defined, a constraint-based method (PC [14]) is applied to each, while mismatches in edge directions are resolved during merging by performing CI tests on residuals and finding v-structures. These two sources are also the ones that provide formal definitions for causal partitioning (definitions 2.1.18 and 2.1.19) in section 2.1.3.

Other approaches assume or construct a tentative graph structure as the starting point. [31] first identify an undirected graph through CI testing, which is then recursively split into overlapping groups by finding sets of nodes that d-separate graph components. Local DAGs for each partition are found using a score based method (MIT [79]). Local graphs are then merged by eliminating directed edges not consistently present across groups and so that existing v-structures remain unchanged. [30] assumes that an undirected "superstructure" containing the true graph is already available. Using community detection algorithms such as greedy modularity, they produce disjoint partitions, which are then expanded into overlapping ones by including adjacent nodes from the full superstructure. Best results were achieved by running the GES [17] and DAGMA [16] score-based algorithms on each partition. The merge phase then joins local graphs, discarding edges not present in the initial superstructure. In an edges appears in all partitions, it is kept as undirected before determining orientation using v-structures. Notably, the experiments in [30] show their method outclassing the one introduced by [74]. Finally, [37] explores a novel direction by incorporating large language models (LLMs) into the divide step. For each variable, an LLM is queried to suggest a set of potential parents, under the assumption that the true parents are contained within the suggested set. This results in a preliminary DAG. The nodes are then divided into overlapping partitions using a greedy search, such that no edges exist between the non-overlapping portions. Within each group, any baseline algorithm may be applied to learn a DAG, with best results obtained using the Hill Climbing [80] score-based algorithm. Par-

Source	Divide	Conquer	Merge
[74]	Cluster (correlation)	PC	Add edges (correlation & CI tests), filter with BIC
[76]	Cluster (correlation & MI)	PC	Add edges (CI tests)
[77]	Cluster (correlation) with merging	PC	Add edges (correlation & CI tests), filter with BIC
[34]	Find d-separating sets	LiNGAM	Solve conflicts using edge ranking
[34]	Low-order CI tests	PC	Solve conflicts using CI tests and v-structures
[31]	Find d-separating sets in undirected graph	MIT	Remove inconsistent edges, keep v-structures
[30]	Community detection on undirected graph	GES, DAGMA	Discard inconsistent edges, orient using v-structures
[37]	Partition an LLM-suggested DAG	HC	Remove edges using BIC

Table 3.1: Summary of methods for each phase in Divide-and-Conquer

tial graphs are then joined, with conflicts in overlapping portions resolved through BIC-based edge elimination, and cycles removed using a similar BIC-guided criterion. Interestingly, [37] provides a comparison between their merging method, a "naïf" approach where the merged graph is just the union of the subgraphs, and the methods in [36], [34] and [31]. Their benchmarking on datasets from the Bayesian Network Repository shows that their BIC-based method for solving conflicts yields the best results.

Different methods for each divide-and-conquer phase are summarized in table 3.1. It is evident that the partitioning process can be performed with two distinct approaches. It can either start (or find) an oriented superstructure (such as in [31]. [81], [30]), or find a causal partitioning from data ([36], [34], [74]). In the latter case, partitioning is estimated by a heuristic, either based on clustering or on low-order CI tests.

Further, not all mentioned methods produce a causal partitioning satisfying the definitions in section 2.1.3. In particular, approaches based on hierarchical clustering actually output disjoint partitions, and thus have to add connections between

partitions during the merge phase.

3.2 Causality and Large Language Models

The potential application and challenges of LLMs in causal inference have been the subject of recent research. Methods mostly rely on injecting *metadata*, such as the names of variables and additional context on the dataset, in the prompts for an LLM.

An influential paper for this study is [82], which systematically tested OpenAI's GPT-3 and two other models on causal discovery tasks. The experiments covered six bayesian network benchmarks of modest size ($n = 2 - 10$ nodes). The models were prompted in natural language with multiple causal queries such as "Does X cause Y?" or "Is there a causal connection between X and Y?", using different wording/phrasing. The models had no access to external knowledge or data. The authors also queried models on abstract reasoning (logical causal chain problems, e.g., "If A causes B and B causes C, does A cause C?") and intuitive physics scenarios (common-sense cause–effect situations such as objects falling off tables). GPT-3 performed best overall, with relatively high accuracy on both abstract reasoning (11/15 correct) and intuitive physics (21/36 correct), though it failed once causal chains grew over six links. In contrast, other models often produced nonsensical or unstable answers. In structure discovery tasks, model behavior varied systematically: GPT-3 tended to underpredict edges, producing overly sparse graphs, while the other models (Luminous, OPT) often overpredicted connections. Models also showed sensitivity to prompt phrasing. "Symmetric" forms like "Are X and Y causally related?" led to ambiguous bidirectional associations, whereas "asymmetric" prompts like "Does X cause Y?" made GPT-3 output more accurate directed edges, with fewer false positives. The authors conclude that GPT-3 shows promising but limited causal competence: it can reproduce fragments of causal knowledge, likely learned from its training corpus ("correlations on top of causation") and answer many commonsense causal questions, however it does not reliably generalize. While such models are not causal discovery tools in the classical sense, they argue that they may serve as useful starting points or "knowledge priors", to be combined with explicit causal inference methods.

More recently, the same authors in [83] formalized why LLMs sometimes appear to succeed at causal inference. They introduced the notion of meta-SCMs: SCMs (see section 2.1) where the variables encode high-level statements about causal relations

(e.g., “altitude causes temperature”) rather than the specific variables themselves. LLMs may answer causal questions correctly not because they infer causality from data, but because during training they have memorized correlations between such causal statements, essentially "correlations" about causality. To test this hypothesis, they repeated the experimental setup of [82]. Their results supported the meta-SCM explanation: GPT-3 and the other models could reproduce known causal facts when these were well-represented in training or external data, but failed when new "reasoning" was required. This reinforces their point that LLMs hold fragments of causal knowledge but are unable to truly perform inference. An interesting contribution was the inclusion of retrieval from the external source ConceptNet [84]: they showed that accuracy improved only if the relevant causal facts are explicitly present in the knowledge base. They used the term “causal parrots” to stress that scaling transformer models alone does not yield genuine causal reasoning. However, their experiments confirmed that RAG (see section 2.2.3) can meaningfully boost performance.

As listed by [85], common approaches to employ LLMs in causal discovery include:

- Pairwise queries, such as those used by [82, 83] as mentioned above and also by PyWhyLLM [86][1]
- "Efficient" search methods to reduce the number of queries, such as the Breadth-First Search (BFS) method by [87][2]
- Combining LLMs with statistical CD methods (see section 2.1.1) [88], [89]
- Agentic systems with access to external knowledge and/or coding tools to perform data analysis, used by [85] themselves, [90], [91], [92]

Overall, results show improvements over baseline methods such as PC, LiNGAM, NOTEARS, and GES on smaller *bnlearn* networks ($n = 5 - 27$), with several works also demonstrating competitive performance on the larger Neuropathic Pain benchmark (see section 2.1.2).

The pairwise approach of PyWhyLLM [86] showed that GPT-3.5 and GPT-4 can recover causal edges with non-trivial accuracy: on a reduced (100 nodes) Neuropathic benchmark, GPT-3.5-turbo achieved an F1 score of 0.68, more than double the random baseline, while GPT-4 achieved the lowest Hamming distance (0.22) on the Arctic sea ice dataset (27 nodes), outperforming NOTEARS, DAG-GNN, and TCDF. The BFS method [87] surpassed GES, PC, NOTEARS, and DAGMA on

[1] Code available on GitHub
[2] Code available on GitHub

Asia (8 nodes) and Child (20 nodes), and is notably the only method claiming to scale effectively to the full Neuropathic (222 nodes), where most classical algorithms become intractable.

[89] showed that GPT-4's prior knowledge augmentation technique improves baseline PC, Exact Search, and DirectLiNGAM on Auto MPG (5 variables), DWD climate (6 variables), and Sachs (11 variables) benchmarks.

Coding agents in [85] outperform PC and DirectLiNGAM on Auto MPG, DWD Climate, and Sachs. The multimodal framework in [91] surpasses both statistical and LLM-only baselines across Auto MPG, DWD Climate, Sachs, Asia, and Child. Finally, the Agentic Stream of Thought (ASoT) proposed by [92], combining GPT-4o, Claude 3.5, and Phi-3 ensembles, outperforms many baselines (PC, GES, NOTEARS, GAE, DAG-GNN) on Auto MPG, DWD Climate, Sachs, Sangiovese (15 nodes), and Neuropathic (100 nodes).

Another major influence for this study, on the practical side, was [88] [3], which introduces CausalCopilot, a chat interface to guide causal discovery with human feedback. Introducing human feedback is an approach gaining popularity in the fields of machine learning and AI, and is commonly referred to as *Human-in-the-loop* [93, 94].

CausalCopilot [88] lets the user start a "conversation" by uploading a dataset and a description, and its key purpose is on performing a complete causal inference pipeline, starting with data exploration, describing variable labels, and statistical tests to determine correlated groups. It then selects an appropriate causal discovery algorithm (such as PC, FCI, XGES, KCI) given the characteristics of the dataset. CausalCopilot hands over key decisions to the user at fixed checkpoints, to double-check its findings or steer the process. For instance, the system asks:

- Whether the user wants to focus on a particular subset of variables
- If the user approves the selected CD algorithm
- If the user wants to edit the discovered causal graph by adding or removing edges

The authors employed synthetic and semi-realistic benchmarks ranging from small graphs with 5-25 nodes up to large-scale networks of 1000 nodes from various domains, including clinical, financial and social network data. They compared performance against standard causal discovery baselines (PC, FCI, GES, DirectLiNGAM)

[3] Code available on GitHub

as well as time-series methods (PCMCI, DYNOTEARS, VARLiNGAM, NOTEARS) and a zero-shot LLM baseline (GPT-4o). While performance is comparable to baselines on simple graphs, CausalCopilot shows improvements on large, noisy, or heterogeneous datasets where baselines often fail to scale. A key limitation is that human interaction occurs at fixed checkpoints, and that the user has to write specific commands or write using a specific format. This allows for limited flexibility, and does not allow for user interaction during the actual discovery process.

3.3 Agentic Systems in Science

Section 2.2.5 introduced the potential of collaboration between multiple LLM instances equipped with external tools, or "agents". Several works exist on the implementation of agentic systems for real-life science and engineering tasks.

A recent example is POPPER [68]. In this framework, a hypothesis is defined as a general statement about the relationship between variables: for instance, “gene GRAP2 regulates IL-2 production.” POPPER adopts an Evaluator–Optimizer pattern (see Section 2.2.5): an "Experiment Design Agent" proposes a concrete testable implication (a sub-hypothesis) of the main hypothesis, and an "Experiment Execution Agent" carries it out. For example, given the GRAP2 hypothesis, the Design agent might suggest checking whether GRAP2 is expressed more strongly in immune tissues than in others. The Execution agent then retrieves a relevant dataset, runs a statistical test, and returns a result. If the result does not provide enough evidence, the Design agent is prompted again. The overall workflow includes several nuanced interactions. The Design agent contains a self-refinement loop: after producing an initial proposal, it critiques and revises it on aspects such as causality, feasibility, and redundancy. Said proposal is then passed to a "Relevance Checker" agent, which verifies that the test is aligned with the original hypothesis, before being handed to the Execution agent. The latter is actually implemented as multiple agents: a "Coding Agent" that selects an appropriate dataset and writes Python code to run the test, an "E-Value Estimation" agent that interprets evidence, finally a "Summarizer" agent that elaborates a conclusion. Benchmarked across domains such as biology, economics, and sociology, POPPER shows how agentic systems can automate systematic validation of scientific claims. Remarkably, in biology it performed on par with PhD-level human experts, completing the task in a fraction of the time. POPPER thus demonstrates how a well-designed agentic system can serve as a practical "substitute" for domain experts in scientific reasoning given a well-defined problem.

Another relevant framework is MetaGPT [95], which focuses on software engineering as a real-world testbed for multi-agent systems. Its design adopts an "assembly line" interaction pattern: specialized agents (Product Manager, Architect, Engineer, QA) interact through structured outputs such as requirement documents, diagrams, and interface specifications. The Architect translates the Product Manager's requirements into system designs, which are then implemented by the Engineer and verified by the QA agent. A self-correction loop allows the Engineer agent to iteratively execute, debug and refine code. MetaGPT aims to mirror human procedures, verifying intermediate results at each step and reducing cascading errors. Agent interaction occurs through a shared message pool, and each agent is able to selectively access just the ones relevant to its role. Its results on the HumanEval [96] and MBPP [97] coding benchmarks reached state-of-the-art code generation performance. MetaGPT also achieved near-perfect completion on complex software development tasks.

As mentioned in section 2.2.3, a line of research and experimentation is that of "Agentic" RAG systems, an example being LangChain's "Local Deep Researcher" [4]. The general principle is that, instead of simply retrieving relevant text passages from a knowledge base using similarity between query and documents, a series of LLMs refine query wording, retrieve passages, and produce a summary. Specifically, Local Deep Researcher searches the web using search engine APIs, such as DuckDuckGo's API. Web page contents are summarized, keeping track of the original source. A "reflection" agent then reads the summary draft: if it is deemed unsatisfactory, another agent writes a more specific query and calls for an additional round of web search; otherwise, it returns the final answer.

Section 2.2.5 stated that agents keep a memory of their interactions. [98] provide a convenient overview of concepts and techniques in memory mechanisms. Memory can be broadly divided into short-term and long-term memory. Short-term memory corresponds to the information accumulated within a single sequence of consecutive interactions, often managed through message histories or caches. Long-term memory stores knowledge that persists across multiple interaction sequences or even multiple tasks, including information from external sources. Design patterns for managing long-term text memory include storing summaries of past interactions and retrieving them through embedding-based similarity, akin to RAG systems.

[4] Code available on GitHub

Chapter 4

Method

This chapter describes the architecture and core principles of a novel method for causal discovery. Following the discussion in section 3.1, each phase of the divide-and-conquer algorithm will be described separately.

4.1 Main Idea

Automating root cause analysis represents a key research interest in manufacturing scenarios, as well as many science and engineering applications. This project introduces an AI assistant intended to aid human experts in causal discovery, leveraging both LLMs' abilities and statistical methods, wrapped in a chat-like user interface.

Inspired by previous research, introduced in sections 3.1 and 2.2.5, the proposed method combines three design patterns:

- Multi-agent collaboration
- Divide-and-Conquer approach
- Human-in-the-Loop

As previously discussed, all three aspects have been proven to be effective in scaling causal discovery on observational data. The original contribution introduced in this study is their combination.

Specifically, the goal is obtaining satisfactory performance on a realistic simulated dataset, the CausalMan benchmark [33] (see section 2.1.2), composed of 186 observed variables of mixed type (both discrete and continuous) and whose underlying process involves complex, non-linear structural relationships, as well as many hidden

confounders.

The system is composed of: an `Explain` module; a `Divide` module, a `Conquer` module, a `Combine` module, a `RAG` module, a `Memory` module.

Overall, the process follows a sequential pattern:

I. The user provides a dataset and (optionally) a description

II. The `Explain` module expands the user-provided metadata

III. The `Divide` module recursively partitions the variables (see section 2.1.3)

IV. For each partition, the `Conquer` module finds a causal graph

V. The `Combine` module combines graphs together, until the final, full result is obtained

Within the four main modules (`Explain`, `Divide`, `Conquer`, `Combine`), LLM agents may query the `RAG` and `Memory` modules, as well as the user.

A key hyperparameter is k, being the maximum allowed size for variable partitions. It is however not used by all methods within `Divide`.

The main frameworks used for building the multi-agent architecture are the `LangChain` [99] and `LangGraph` [100] python libraries.

The following sections will detail the functioning of each module, starting with the two "helper" components.

4.2 RAG and Human-in-the-Loop

An ideal AI assistant should efficiently acquire external information to keep its answers grounded in reliable sources (see sections 2.2.2 and 2.2.3). Every agent in the main modules may, at any point, gather external knowledge either via Retrieval-Augmented Generation (RAG) or via human interaction (Human-in-the-Loop). Both are discussed here as they serve similar purposes and because, from a practical point of view, both options are provided to each LLM instance as tools (see section 2.2.2).

For enabling **Human-in-the-Loop**, agents may call a function that queries the user for additional information, printing a natural language question either via terminal or via chat interface. For instance, an agent may ask for the definition of a cryptic label, more detailed context or to confirm a causal relationship. As mentioned, this

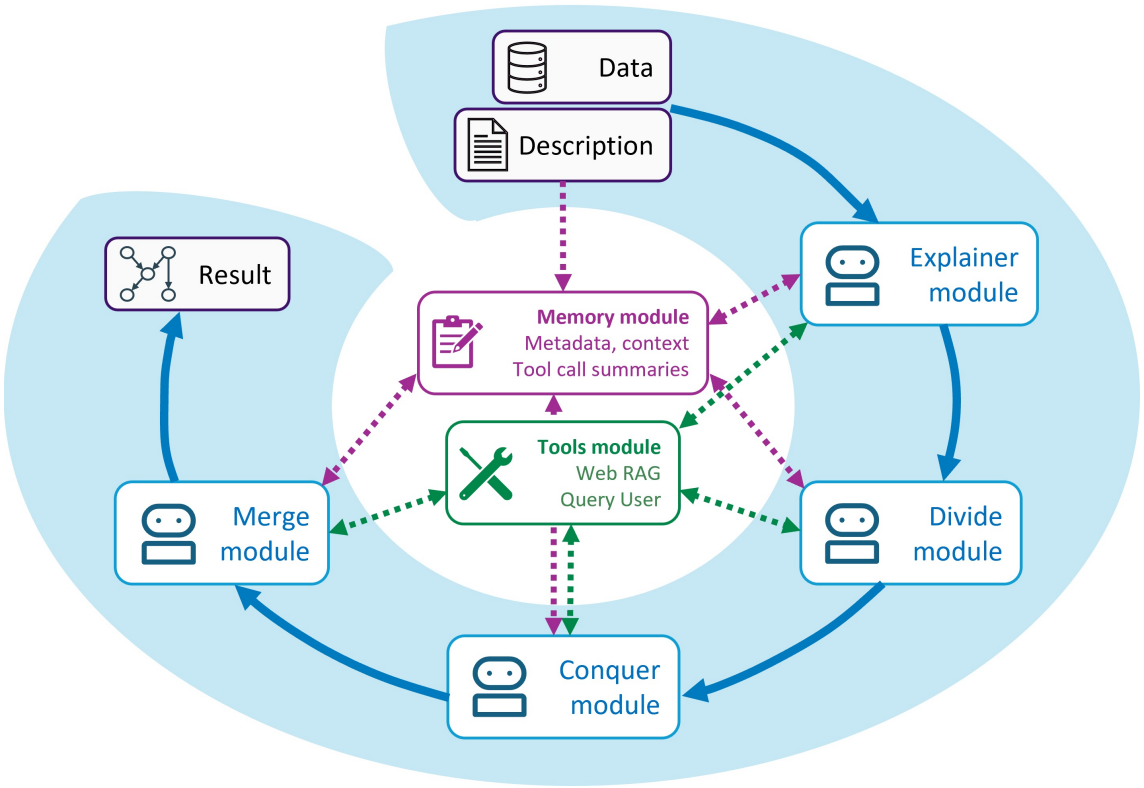

Figure 4.1: Overview of the proposed method and its main components

interaction is made on-demand as a tool call. The user can freely type an answer, which is directly sent to the agent as a tool call output.

Similarly, agents have the capacity to dynamically invoke an **Agentic RAG** system on demand. This is implemented as a tool wrapping of LangChain's "Local Deep Researcher" [1], introduced in section 3.3, itself equipped with DuckDuckGo's search API. Starting from a natural language query, Local Deep Researcher can autonomously perform multiple rounds of web search, refine the query and summarize results.

4.3 Memory

The memory mechanism is implemented as a data structure M designed to store metadata and summaries of Agents' past interactions with the user or the `RAG` module, as to avoid redundant questions or web searches.

It holds the following:

- `domain`: a string containing the dataset's domain D_{domain}, for example:

[1] GitHub

```
"domain": "Manufacturing and press-fitting"
```

- `general_description`: a string containing the overall description of the dataset, $D_{description}$, for example:

```
"general_description": "The dataset contains sensor measurements
↪  and quality control flags from a production line. The
↪  manufacturing process involves press-fitting of..."
```

- `variable_descriptions`: a dictionary containing the label of each variable in the dataset D_{labels}, and a corresponding detailed description, for example:

```
"variable_descriptions": {
  "PF_M1_T1_Force": "Press-fitting force applied by machine 1 on
  ↪  bore 1..."
}
```

- `tool_call_summaries`: a dictionary of dictionaries where past tool calls are stored under unique ids, for example:

```
"tool_call_summaries": {
  "1": {
    "variables_of_focus": ["PF_M1_T1_Force", "PF_M1_T1_Fmax"...],
    "summary": "Maximum force during press fitting may be
    ↪  influenced by displacement and average force...",
    "embedding": [-0.057, 0.762, ...]
  }
}
```

Prior to discussing memory memory management, it is important to note that the main modules (`Explain`, `Divide`, `Conquer`, `Combine`) are designed to be executed on a subset of the full dataset D. For example, if $\mathbf{V} = D_{labels}$ is the full set of variables, an iteration of `Divide` may run on a smaller partition $\mathbf{P} \subseteq \mathbf{V}$. This aspect will be further detailed in their respective sections. What is relevant in terms of memory management is that, at each point during the execution of the main modules, the

system keeps track of what variable set $\mathbf{P}$ the module, and thus any specific LLM agent within it, is processing.

Memory writing occurs in two ways. The dataset's metadata, namely `domain` and `general_description`, are initialized with the user-provided description (D_{domain} and $D_{description}$), while `variable_descriptions` gets initialized as an empty dictionary. During the execution of the `Explainer` module, `general_description` and `variable_descriptions` are progressively updated. Tool call outputs, whether originating from human messages or from the `RAG` module, are processed in the following manner. The raw string output [2] is first summarized by a *Summarizer* LLM $\mathcal{S}$, guided by few-shot prompting. In addition to the textual summary, $\mathcal{S}$ also produces a list of variable labels (among the ones in $\mathbf{P}$) to which the content is most relevant. Employing a type of *keyword augmentation* for embeddings inspired by [101], the summary and its associated labels are concatenated into a single string, which is then passed as input to an embedding LLM $\mathcal{E}$. The resulting embedding vector is subsequently stored within the memory entry.

Memory reading is also handled in two different modes. Metadata is injected into the prompts for each agent in the main modules. Specifically, `domain` and `general_description` are inserted into all downstream prompts.
`variable_descriptions` are selected based on what variable subset $\mathbf{P}$ is being processed by the LLM at invoking time. Tool call summaries are retrieved using embedding similarity: before invoking an agent, the variable labels within the subset $\mathbf{P}$ are inserted in a retrieval query [3].
$\mathcal{E}$ then embeds the query, and cosine similarities between query and summary embeddings are used to retrieve summaries according to a similarity threshold. If any matches are found, the corresponding entries T_M are inserted into the current agent's prompt.

4.4 Explain

Prior to the `Divide-Conquer-Combine` modules, an *Explainer* LLM agent $\mathcal{X}$ gathers additional information about the task and dataset context to expand the user-provided metadata. By doing so, it decreases the ambiguity about the meaning of each variable.

[2] e.g.: `"Studies show that force measurements in manufacturing are influenced by tolerances..."`

[3] e.g.: `"Retrieve passages relevant to these variables: PF_M1_T2_Force, PF_M1_T2_Fmax"`

$\mathcal{X}$ operates in two modes. For both, $\mathcal{X}$ employs *ReAct*-style prompting (see section 2.2.4, albeit with different examples depending on the mode.

First, all user-provided metadata, i.e. domain D_{domain}, variable labels D_{labels} and description $D_{description}$ are provided to the agent. Next, $\mathcal{X}$ outputs an expanded dataset description $D^{\mathcal{X}}_{description}$:

$$D^{\mathcal{X}}_{description} \leftarrow \mathcal{X}(\mathbf{p}_{\mathcal{X},1}, D_{domain}, D_{description}, D_{labels}) \tag{4.1}$$

$\mathbf{p}_{\mathcal{X},1}$ is the set of fixed prompts containing general directions for the agent and *ReAct*-style examples (see section 2.2.4) for the first mode. $D_{description}$ and D_{labels} are inserted into a separate prompt template.

In the second mode, $\mathcal{X}$ is prompted to provide a detailed definition of each variable label:

$$D^{\mathcal{X}}_{labels} = \mathcal{X}\left(\mathbf{p}_{\mathcal{X},2}, D_{domain}, D^{\mathcal{X}}_{description}, D_{labels}, T_M\right) \tag{4.2}$$

$\mathbf{p}_{\mathcal{X},2}$ are the fixed prompts, with examples, for the second *Explainer* mode, and T_M are past tool call summaries retrieved from M. Pragmatically, variables in D_{labels} are divided into batches of $2k$-size, to avoid context window limits.

The updated metadata $D^{\mathcal{X}}_{description}$ and $D^{\mathcal{X}}_{labels}$ are finally stored in the memory M, under `general_description` and `variable_descriptions` fields respectively.

4.5 Divide

The *Divide* phase recursively partitions the variable set $\mathbf{V}$ with the aim of grouping causally related variables together.

Theoretical definitions on *Causal Partitioning* were discussed in section 2.1.3, the bottom line being that obtaining a valid one is key for not losing information. Therefore, this section introduces multiple alternative methods within the `Divide` module.

The first strategy (referred to as *Meta-Agents*) is to rely solely on metadata, and thus on a partitioning suggested by LLM Agents. Two other, hybrid strategies, combine data and metadata as follows. LLMs are first used to generate a tentative clustering of variables: either by querying agents, as in Meta-Agents, or using an embedding-based clustering. Then, similarly to CAPA [36] (see section 3.1), low-order CI tests are applied to construct an adjacency matrix. The clustering provided by the LLMs serves to restrict the search space of these tests. Following [30] (also discussed in Section 3.1), the resulting structure is partitioned into dis-

joint communities via a community detection algorithm. These communities are then expanded by adding neighboring nodes from the full adjacency matrix. These hybrid strategies are denoted *Hybrid-Agents* and *Hybrid-Embeddings*, depending on the LLM clustering method.

4.5.1 Meta-Agents Method

This is carried out by two LLM agents, in an Evaluator-Optimizer pattern with a single iteration (see section 2.2.5): the Divide-hypothesis agent $\mathcal{D}_h$ and the Divide-critic $\mathcal{D}_c$ agent. Both agents leverage the information gathered by the explainer agent to estimate a causal partitioning of the variable set $\mathbf{V}$.

Formally, given a set of variables $\mathbf{P}_i \subseteq \mathbf{V}$, the Divide-hypothesis agent $\mathcal{D}_{hyp}$ first proposes a possible partitioning,

$$\{\mathbf{P}_{i,1}, \ldots, \mathbf{P}_{i,N}\} \leftarrow \mathcal{D}_h \left(\mathbf{p}_{\mathcal{D}h}, \mathbf{P}_i, D_{domain}, D^{\mathcal{X}}_{description}, D^{\mathcal{X}}_{labels} \cap \mathbf{P}_i, T_M\right) \tag{4.3}$$

where $\mathbf{P}_{i,j}$ denotes the j-th partition of $\mathbf{P}_i$, such that $\bigcup_j \mathbf{P}_{i,j} = \mathbf{P}_i$. $\mathbf{p}_{\mathcal{X}h}$, is the set of fixed prompts for $\mathcal{D}_h$ (containing general instructions and *ReAct* style examples). D_{domain} is the dataset's domain. $D^{\mathcal{X}}_{description}$ is the expanded dataset description and $D^{\mathcal{X}}_{labels} \cap \mathbf{P}_i$ is the subset of variable definitions corresponding to the labels in $\mathbf{P}_i$, both stored in M. T_M are past tool call summaries retrieved from M. Different partitions can overlap, i.e. $\mathbf{P}_{i,j} \cap \mathbf{P}_{i,l}$ might be non-empty for some j and l.

Following, a $\mathcal{D}_c$ is tasked to review the partitioning and apply corrections if necessary:

$$\{\tilde{\mathbf{P}}_{i,1}, \ldots, \tilde{\mathbf{P}}_{i,N}\} \leftarrow \mathcal{D}_c \left(\mathbf{p}_{\mathcal{D}c}, \{\mathbf{P}_{i,1}, \ldots, \mathbf{P}_{i,K}\}, D_{domain}, D^{\mathcal{X}}_{description}, D^{\mathcal{X}}_{labels} \cap \mathbf{P}_i, T_M\right) \tag{4.4}$$

$\mathbf{p}_{\mathcal{X}c}$, is the set of fixed prompts for $\mathcal{D}_c$ (with *ReAct* style examples). A single invocation of $\mathcal{D}_h$ and $\mathcal{D}_c$ is considered a *Divide* step.

Partitions are stored as nodes in a tree-like data structure, a *Partition Tree*, where $\tilde{\mathbf{P}}_{i,1}, \ldots, \tilde{\mathbf{P}}_{i,N}$ are children of $\mathbf{P}_i$, with the root node being the full set $\mathbf{V}$. The causal partitioning process proceeds recursively: for any leaf node $\mathbf{P}_i$ in the Partition Tree, such that $|\mathbf{P}_i| > k$, a *Divide* step is executed again. At this stage, the agents $\mathcal{D}h$ and $\mathcal{D}c$ decide whether to further split the partition $\mathbf{P}_i$; if they do not, the current level of granularity is considered sufficient for causal discovery.

Prompts $\mathrm{p}_{\mathcal{D}h}$ and $\mathrm{p}_{\mathcal{D}c}$ include instructions on dividing variables into a causal partitioning. In particular, they contain actionable directions based on definition 2.1.18,

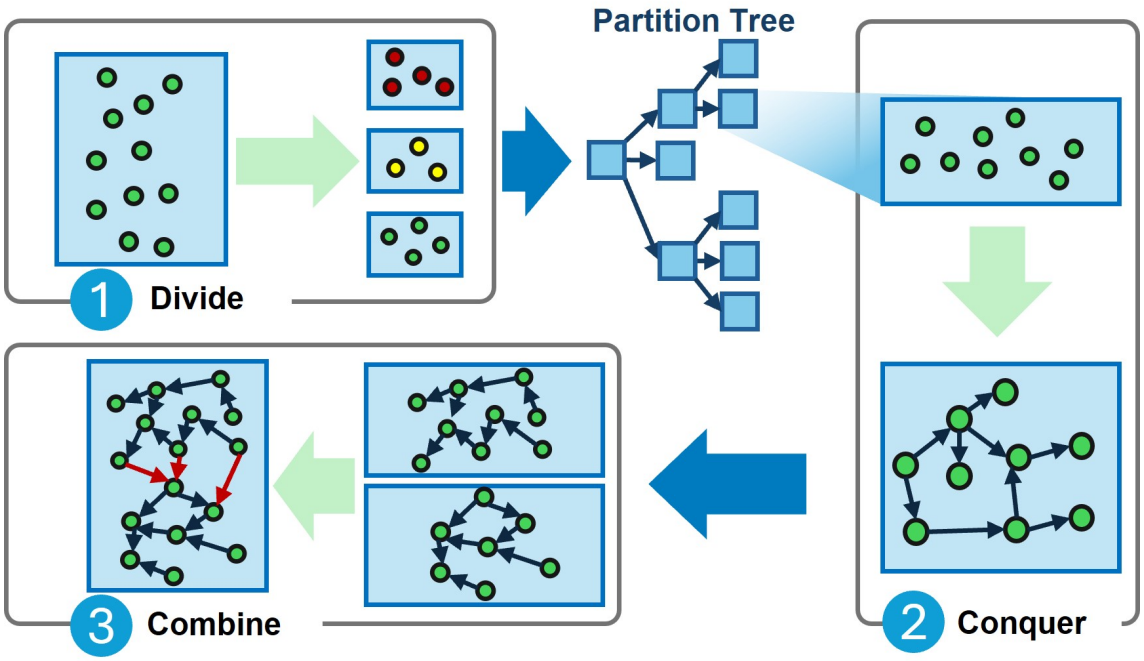

Figure 4.2: Overview of the *Divide-and-Conquer* paradigm, displaying how hierarchical partitions are stored in a *Partition Tree* structure after the *Divide* phase

suggesting to output overlapping groups such that potentially connected variable pairs are "together" in at least one group.

4.5.2 Hybrid-Agents Method

This method employs an algorithm for iterative CI tests (similar to CAPA [36]), a community detection algorithm combined with "causal expansion" (as in [30]), and the $\mathcal{D}_h$ and $\mathcal{D}_c$ agents pair. Referring back to equations 4.3, 4.4 and the definition of *Partition Tree* from the previous section, the algorithm below provides an overview of the process.

In Algorithm 1, *Divide* agents $\mathcal{D}_h$ and $\mathcal{D}_c$ are first called to suggest a clustering of variables in $\mathbf{V}$, $\mathcal{C}$, rather than directly update the Partition Tree.

The data-driven algorithm (lines 8-19 in algorithm 1) then acts on leaf (child-less) nodes $\mathbf{P} \subseteq \mathbf{V}$ in the Partition Tree, including the root node, i.e. V itself. For any leaf node $\mathbf{P}$ with more than k variables, the routine `CI_CD_partitioning` (details in 2) is called: its job is first to update the adjacency matrix A by performing CI tests of order t between pairs of variables within $\mathbf{P}$. If the specified order t is greater than 0, it uses the clustering $\mathcal{C}$ to restrict the CI tests to only pairs of variables belonging to different clusters, and such that the conditioning set is also within the

Algorithm 1 Hybrid-Agents Overview

Input: dataset D, variable set $\mathbf{V}$, t_{max}, k
AddNode(PartitionTree, $\mathbf{V}$) ▷ Root of Partition Tree
Attempts $\leftarrow \{\}$ ▷ Dictionary: set $\mathbf{P} \mapsto$ CI test order t
$A \leftarrow \mathbf{1}_{n \times n}$ ▷ Adjacency matrix, $n := |\mathbf{V}|$
$\{\mathbf{C}_1, \ldots, \mathbf{C}_N\} \leftarrow \mathcal{D}_h(\mathbf{V})$
$\{\tilde{\mathbf{C}}_1, \ldots, \tilde{\mathbf{C}}_N\} \leftarrow \mathcal{D}_c(\mathbf{V})$
$\mathcal{C} \leftarrow \{\tilde{\mathbf{C}}_1, \ldots, \tilde{\mathbf{C}}_N\}$ ▷ Clusters from agents
for all leaf nodes $\mathbf{P}$ in PartitionTree with $|\mathbf{P}| > k$ **do**
    if $\mathbf{P}$ in Attempts **then**
        $t \leftarrow$ Attempts$[\mathbf{P}] + 1$
    else
        $t \leftarrow 0$
    if $t \leq t_{max}$ **then**
        A, $\{\mathbf{P}_1, \ldots, \mathbf{P}_N\} \leftarrow$ CI_CD_PARTITIONING($D_{\mathbf{P}}$, $\mathbf{P}$, A, t, $\mathcal{C}$)
        if $N > 1$ **then** ▷ If more than one partition
            for all partitions $\mathbf{P}_i$ in $\{\mathbf{P}_1, \ldots, \mathbf{P}_N\}$ **do**
                Attempts$[\mathbf{P}_i] \leftarrow t$
                AddNode(PartitionTree, $\mathbf{P}_i$)
        Attempts$[\mathbf{P}] \leftarrow t$
for all leaf nodes $\mathbf{P}$ in PartitionTree with $|\mathbf{P}| > k$ **do**
    $\{\mathbf{P}_1, \ldots, \mathbf{P}_N\} \leftarrow \mathcal{D}_h(\mathbf{P})$ ▷ Invoke agents on large partitions
    $\{\tilde{\mathbf{P}}_1, \ldots, \tilde{\mathbf{P}}_N\} \leftarrow \mathcal{D}_c(\mathbf{P})$
    for all partitions $\mathbf{P}_i$ in $\{\tilde{\mathbf{P}}_1, \ldots, \tilde{\mathbf{P}}_N\}$ **do**
        AddNode(PartitionTree, $\mathbf{P}_i$)
return PartitionTree

union of the two clusters. A community detection algorithm (Greedy Modularity [102]) is then called on the subset of A corresponding to the variables in $\mathbf{P}$ ($A_{\mathbf{P}}$). Greedy Modularity splits $\mathbf{P}$ into disjoint communities ($\mathbf{P}_i$).

Communities undergo an "expansion" process, where for each community $\mathbf{P}_i$ an expansion set $\mathbf{E}$ is constructed, containing all nodes X_l such that they are adjacent (in A) to some node X_k within $\mathbf{P}_i$. If all expanded communities $\tilde{\mathbf{P}}_i$ are smaller than the starting variable set $\mathbf{P}$, then the partitioning is successful and they get added to the Partition Tree as children of $\mathbf{P}$. On the contrary, if said partitioning fails, the order of the CI tests t is stored, so that the same partition can be processed again using a higher order.

After the data-driven phase is completed, if any partitions are still over the k size threshold, the LLMs $\mathcal{D}_h$ and $\mathcal{D}_c$ are invoked again (lines 20-24) to try and split them further.

The standard `networkx` implementation of Greedy Modularity [103] is used at line 8 of algorithm 2. The chonsen CI test (lines 4 and 13 of algorithm 3), is the χ^2 test from the `causal-learn` library [104].

Algorithm 2 Partitioning with CI tests and community detection

1: **procedure** CI_CD_PARTITIONING(dataset subset $D_{\mathbf{P}}$, variable set $\mathbf{P}$, adjacency matrix A, CI test order t, variable clustering $\mathcal{C}$)
2: **if** $t = 0$ **then**
3: $A' \leftarrow$ CI_TESTS(D, $\mathbf{P}$, t)
4: **if** $t > 0$ **then**
5: $A' \leftarrow$ CI_TESTS_RESTRICTED(D, $\mathbf{P}$, t, $\mathcal{C}$)
6: **if** $A' \neq A$ **then**
7: $A \leftarrow A'$
8: $\{\mathbf{P}_1, \ldots, \mathbf{P}_N\} \leftarrow$ GREEDYMODULARITY($A_{\mathbf{P}}$)
9: **for** $\mathbf{P}_i \in \{\mathbf{P}_1, \ldots, \mathbf{P}_N\}$ **do** ▷ Community expansion
10: $\mathbf{E} \leftarrow \emptyset$
11: **for** $X_k \in \mathbf{P}_i$ **do**
12: **if** $A_{k,l} = 0$ for some $X_l \notin \mathbf{P}_i$ **then**
13: $\mathbf{E} \leftarrow \mathbf{E} \cup \{X_l\}$
14: $\tilde{\mathbf{P}}_i \leftarrow \mathbf{P}_i \cup \mathbf{E}$
15: **if** $|\tilde{\mathbf{P}}_i| < |\mathbf{P}|$ for all $\tilde{\mathbf{P}}_i \in \{\tilde{\mathbf{P}}_1, \ldots, \tilde{\mathbf{P}}_N\}$ **then** ▷ Check sizes
16: **return** A, $\{\tilde{\mathbf{P}}_1, \ldots, \tilde{\mathbf{P}}_N\}$
17: **else**
18: **return** A, $\{\}$

Algorithm 3 Conditional Independence testing routines

procedure CI_TESTS(dataset D, variable set $\mathbf{P}$, test order t)
  for $X, Y \in \mathbf{P}$ **do**
    for $\mathbf{Z} \subseteq \mathbf{P} \setminus \{X, Y\}$ such that $|\mathbf{Z}| = t$ **do**
      $p_{val} \leftarrow$ CI_TEST(D_X, D_Y, $D_{\mathbf{Z}}$)
      if $p_{val} > \tau$ **then** ▷ p-value threshold hyperparameter
        $A_{X,Y} \leftarrow 0$
  return A

procedure CI_TESTS_RESTRICTED(dataset D, variable set $\mathbf{P}$, test order t, clusters $\mathcal{C}$)
  for $C_1 \in \mathcal{C}$ **do**
    for $C_1 \in \mathcal{C}$, $C_2 \neq C_1$ **do**
      for $X \in C_1$, $Y \in C_2$ **do**
        for $\mathbf{Z} \subseteq C_1 \cup C_2 \setminus \{X, Y\}$ such that $|\mathbf{Z}| = t$ **do**
          $p_{val} \leftarrow$ CI_TEST(D_X, D_Y, $D_{\mathbf{Z}}$)
          if $p_{val} > \tau$ **then** ▷ p-value threshold hyperparameter
            $A_{X,Y} \leftarrow 0$
  return A

4.5.3 Hybrid-Embeddings Method

Similarly to the Hybrid-Agents method, an LLM-based clustering is employed to restrict CI tests of order higher than 1. However, instead of querying agents to suggest a partitioning, clustering is performed on text embeddings.

First, inspired by the keyword augmentation technique described in [101], a *Keywords* agent $\mathcal{K}$ is invoked for each variable `label` and `description` in $D^{\mathcal{X}}_{labels}$:

$$\texttt{keywords} \leftarrow \mathcal{K}\left(\mathrm{p}_{\mathcal{K}}, D_{domain}, D^{\mathcal{X}}_{description}, \{\texttt{label}, \texttt{description}\}, T_M\right) \tag{4.5}$$

Similarly to previously introduced agents, $\mathrm{p}_{\mathcal{K}}$ contains fixed instructions with *ReAct*-style examples and T_M are past tool call summaries retrieved from M. $\mathcal{K}$ is prompted to generate keywords or *keyphrases* related to the variable label and descriptions. Said keywords encode context on the dataset, or act as "tags" useful to group causally related variables together. For example:

$$\mathcal{K}\left(\{\texttt{"PF\_M1\_T1\_Force"}, \texttt{"Pressing force measurement at bore 1..."}\}\right)$$
$$\downarrow$$
$$\{\texttt{"Step 1"}, \texttt{"Machine 1"}, \texttt{"Press Fitting Tolerances"}, \texttt{"Quality Control"}\} \tag{4.6}$$

The `keywords` and `label+description` strings are fed separately to the embedding model $\mathcal{E}$: $\vec{e}_{kw} \leftarrow \mathcal{E}(\texttt{keywords})$ and $\vec{e}_{ld} \leftarrow \mathcal{E}(\texttt{label: description})$. Following the example above this looks like:

$$[0.023, -0.039, ...] \leftarrow \mathcal{E}(\texttt{"PF\_M1\_T1\_Force: Pressing force measurement..."})$$
$$[-0.314, 0.556, ...] \leftarrow \mathcal{E}(\texttt{"Step 1, Machine 1, Press Fitting Tolerances, ..."}) \tag{4.7}$$

The two embedding vectors $\vec{e}_{kw}$, $\vec{e}_{ld}$ are then concatenated into a single 1-d vector $\vec{e}_c$ of doubled dimension. After repeating this embedding step for each `{label, description}`, the stacked set of concatenated vectors $\vec{\mathbf{E}} = [\vec{e}_{c,1}, \ldots, \vec{e}_{c,n}]$ is used as input for a standard agglomerative clustering algorithm from the `sklearn` library, obtaining a set of clusters $\mathcal{C} = \{\mathbf{C}_1, \mathbf{C}_2, ...\}$. The overall process in summarized in algorithm 4.

After this clustering phase, the method follows the same data-driven procedure described in the Hybrid-Agents algorithm 1, lines 8-19. In this method, the final partitioning step done by the *Divide* agents $\mathcal{D}_c$ and $\mathcal{D}_c$ (lines 20-24 in algorithm 1) is skipped.

Algorithm 4 Keyphrase-augmented clustering on embeddings

procedure EMBEDDINGCLUSTERING(dataset domain D_{domain}, dataset description $D^{\mathcal{X}}_{description}$, variable labels $D^{\mathcal{X}}_{labels}$)
  $\vec{\mathbf{E}} \leftarrow [\,]$
  for all {`label`, `description`} in $D^{\mathcal{X}}_{labels}$ **do**
    $\texttt{keywords} \leftarrow \mathcal{K}\left(\mathbf{p}_{\mathcal{K}}, D_{domain}, D^{\mathcal{X}}_{description}, \{\texttt{label}, \texttt{description}\}\right)$
    $\vec{e}_{ld} \leftarrow \mathcal{E}(\texttt{label: description})$
    $\vec{e}_{kw} \leftarrow \mathcal{E}(\texttt{keywords})$
    $\vec{e}_c \leftarrow \vec{e}_{kw} \oplus \vec{e}_{ld}$ ▷ Concatenation in $\mathbb{R}^{2d}$
    Append $\vec{e}_c^{\,\mathsf{T}}$ as a row in $\vec{\mathbf{E}}$
  $\mathcal{C} \leftarrow$ AGGLOMERATIVECLUSTERING($\vec{\mathbf{E}}$)
  return $\mathcal{C}$

4.6 Conquer

The previous section introduced three original methods for recursively partitioning the set $\mathbf{V}$ composed of $n = |\mathbf{V}|$ variables. It also mentioned how partitions are stored in a tree-like data structure, a *Partition Tree*, whose leaf nodes contain the smaller, potentially overlapping, sets of variables $\mathbf{P} \subset \mathbf{V}$, such that $|\mathbf{P}| \leq k < n$ [4].

[4] Though this condition is not guaranteed, as will be evident in section 5

The `Conquer` module is called on each leaf partition and executes the actual Causal Discovery, yielding either a directed mixed graph or a PAG, depending on the chosen method (refer to section 2.1.1).

Running any standard CD algorithms is a viable option in this phase, thanks to the reduced number of variables. In this study, two LLM-driven CD methods are benchmarked, comparing them to baseline algorithms in section 5. One method just relies on LLM agents (*Agents)*, and thus on metadata, and another method takes agents' suggestions as prior knowledge for a constraint based algorithm (*Agents-FCI*).

4.6.1 Agents

Similarly to the *Meta-Agents* method in the `Divide` module, a pair of *Hypothesis-Critic* agents, $\mathcal{C}_h$ and $\mathcal{C}_c$ suggest a local causal graph G_i, in the form of a list of directed edges $\mathbf{E}_i$ for each leaf partition $\mathbf{P}_i$:

$$\begin{aligned} G_i(\mathbf{P}_i, \mathbf{E}_i) &\leftarrow \mathcal{C}_h\left(\mathbf{p}_{c,h}, \mathbf{P}_i, D_{domain}, D^{\mathcal{X}}_{description}, D^{\mathcal{X}}_{labels} \cap \mathbf{P}_i, T_M\right) \\ \tilde{G}_i(\mathbf{P}_i, \tilde{\mathbf{E}}_i) &\leftarrow \mathcal{C}_c\left(\mathbf{p}_{c,c}, \mathbf{P}_i, D_{domain}, D^{\mathcal{X}}_{description}, D^{\mathcal{X}}_{labels} \cap \mathbf{P}_i, G_i, T_M\right) \end{aligned} \tag{4.8}$$

where $\mathbf{p}_{c,h}$ and $\mathbf{p}_{c,c}$ are the fixed *ReAct*-style prompts for both agents. T_M are relevant tool call summaries retrieved from M. Again following an single-iteration evaluator-optimizer pattern, the critic $\mathcal{C}_c$ is asked to review the directed edges G_i proposed by $\mathcal{C}_h$. From pragmatic observations during the system's testing, $\mathbf{p}_{c,h}$ was phrased as to encourage the LLM to suggest (in a sense "overestimate") even weak causal connections, so that G_i is not "too" sparse. Instead, $\mathbf{p}_{c,c}$ asks to reason on whether the proposed connections are valid at an interventional and counterfactual level.

$\mathcal{C}_h$ and $\mathcal{C}_c$ are prompted to just output pairs of variable labels corresponding to directed edges. The resulting G_i and $\tilde{G}_i$ only contains arrow endpoints. The prompts $\mathbf{p}_{c,h}$, $\mathbf{p}_{c,c}$ do not explicitly forbid bi-directed edges or cycles. Thus, formally, G_i and $\tilde{G}_i$ are directed mixed graphs, where edges $\mathbf{E}_i$, $\tilde{\mathbf{E}}_i$ are either directed $X \longrightarrow Y$, or bi-directed $X \longleftrightarrow Y$.

4.6.2 Agents-FCI

In this variant, the directed edges $\tilde{G}_i$ from the $\mathcal{C}_h$, $\mathcal{C}_c$ pair are used to bootstrap the FCI algorithm [15, 19]. Under the assumption that the edges in $\tilde{G}_i$ are grounded in

domain knowledge, within FCI they are set as valid. In equation form:

$$\begin{aligned} G_i(\mathbf{P}_i, \mathbf{E}_i) &\leftarrow \mathcal{C}_h\left(\mathbf{p}_{c,h}, \mathbf{P}_i, D_{domain}, D^{\mathcal{X}}_{description}, D^{\mathcal{X}}_{labels} \cap \mathbf{P}_i, T_M\right) \\ \tilde{G}_i(\mathbf{P}_i, \tilde{\mathbf{E}}_i) &\leftarrow \mathcal{C}_c\left(\mathbf{p}_{c,c}, \mathbf{P}_i, D_{domain}, D^{\mathcal{X}}_{description}, D^{\mathcal{X}}_{labels} \cap \mathbf{P}_i, G_i, T_M\right) \\ \overline{G}_i(\mathbf{P}_i, \overline{\mathbf{E}}_i) &\leftarrow \texttt{FCI}(D_{Pi}, \tilde{G}_i) \end{aligned} \tag{4.9}$$

where, D_{Pi} indicates the subset of the dataset D containing only the variables in the partition $\mathbf{P}_i$, and $\tilde{E}_i \subseteq \overline{E}_i$.

FCI is designed to handle causal insufficiency, and thus outputs a PAG (see section 2.1.1). As such, the edges in $\overline{\mathbf{E}}_i$ may be directed , bi-directed, or feature undefined endpoints. The FCI implementation from the `causal-learn` library is used, equipped with χ^2 as its independence test type.

4.7 Combine

After the `Divide` module has been executed on each leaf partition $\mathbf{P}_i$ in the Partition Tree, the resulting causal graphs of sibling nodes are merged together, pruning the Tree until only the root node $\mathbf{V}$ remains, thus obtaining the full result.

Given $N \geq 2$ local causal graphs $G_1(\mathbf{P}_1, \mathbf{E}_1), ..., G_N(\mathbf{P}_N, \mathbf{E}_N)$ the `Combine` module acts in two phases:

I. Solve inconsistencies by edge elimination

II. Add bridging edges between local graphs

These are detailed in the following sections, while algorithm 5 provides an overview.

4.7.1 Eliminating extra edges

Similarly to [31], conflicts between groups are solved by removing *extra* edges in the overlapping portions of the input graphs $G_1, ..., G_N$.

An illustrative example of what an *extra* edge is the following. Suppose $N = 2$ and $\mathbf{P}_1 \cap \mathbf{P}_2 = \{X, Y\}$. Suppose $X \longrightarrow Y \in \mathbf{E}_1$, but $X \longrightarrow Y \notin \mathbf{E}_2$. The directed edge $X \longrightarrow Y$ is "extra" in the sense that is not consistently present in both G_1 and G_2. In such a case, $\mathbf{P}_2$ contains a mediator Z between X and Y, but $\mathbf{P}_1$ does not: thus the `Conquer` module acting on $\mathbf{P}_2$ added $X \longrightarrow Z, Z \longrightarrow Y$ to $\mathbf{E}_2$.

Similar cases occur for bi-directed edges or undefined endpoints: as mentioned in section 2, these represent ambiguities in the CD process due to hidden confounders.

Suppose now that $X \longleftrightarrow Y \in \mathbf{E}_1$ but $X \longleftarrow Z, Z \longrightarrow Y \in \mathbf{E_2}$. In this situation $X \longleftrightarrow Y$ counts as two extra directed edges in G_1 ($X \longleftarrow Y$ and $X \longrightarrow Y$), which are explained by a confounder Z that is only present in G_2.

4.7.2 Adding bridging edges

As a way to contrast a sub-optimal causal partitioning, the `Combine` module also includes an agents-based heuristic to add edges connecting local graphs.

Similarly to the `Divide` and `Conquer` modules, another Hypothesis-Critic agent pair is used to suggest a list of directed edges. Given $G_1(\mathbf{P}_1, \mathbf{E}_1), ..., G_N(\mathbf{P}_N, \mathbf{E}_N)$ causal graphs, the nodes in each partition $\mathbf{P_i}$ are filtered to only keep the disjoint component from all other partitions: $\overline{\mathbf{P}}_i \subseteq \mathbf{P_i}$, such that $\overline{\mathbf{P}}_i \cap \mathbf{P}_j = \emptyset$ for all $j \neq i$. This step yields set of disjoint sibling partitions $\overline{\mathcal{P}} = \{\overline{\mathbf{P}}_1, \ldots, \overline{\mathbf{P}}_N\}$. Then, two merging agents $\mathcal{M}_h$ and $\mathcal{M}_c$ are invoked:

$$\begin{aligned} \mathbf{E}_c &\leftarrow \mathcal{M}_h \left(\mathbf{p}_{m,h}, D_{domain}, D^{\mathcal{X}}_{description}, D^{\mathcal{X}}_{labels} \cap \overline{\mathcal{P}}, \overline{\mathcal{P}}, T_M\right) \\ \tilde{\mathbf{E}}_c &\leftarrow \mathcal{M}_h \left(\mathbf{p}_{m,c}, D_{domain}, D^{\mathcal{X}}_{description}, D^{\mathcal{X}}_{labels} \cap \overline{\mathcal{P}}, \overline{\mathcal{P}}, \mathbf{E}_c, T_M\right) \end{aligned} \tag{4.10}$$

where $\mathbf{p}_{m,h}$ and $\mathbf{p}_{m,c}$ are again fixed *ReAct*-style prompts, $D^{\mathcal{X}}_{labels} \cap \overline{\mathcal{P}}$ is the subset of variable labels and associated descriptions (stored in memory) corresponding to the variables within the disjoint partitions $\mathbf{P}_i$ within $\overline{\mathcal{P}}$

Algorithm 5 Two-phase merging of causal graphs

procedure COMBINE(`Partition Tree`, graphs $\{G_1(\mathbf{P}_1, \mathbf{E}_1), ..., G_N(\mathbf{P}_N, \mathbf{E}_N)\}$)
  for all $\mathbf{E}_i \in \{\mathbf{E}_1, \ldots, \mathbf{E}_N\}$ **do**
    for all $X \longrightarrow Y \in \mathbf{E}_i$ **do**
      for all $\mathbf{E}_i \in \{\mathbf{E}_1, \ldots, \mathbf{E}_N\}$, $j \neq i$ **do**
        if $\{X, Y\} \subseteq \mathbf{P_j}$ **and** $X \longrightarrow Y \notin \mathbf{E}_j$ **then**
          $\mathbf{E}_i \leftarrow \mathbf{E}_i \setminus \{X \longrightarrow Y\}$

  $\overline{\mathcal{P}} \leftarrow \emptyset$
  for $\mathbf{P}_i \in \{\mathbf{P}_1, \ldots \mathbf{P}_N\}$ **do**
    $\mathbf{O}_i \leftarrow \emptyset$
    for all $\mathbf{P}_j \in \{\mathbf{P}_1, \ldots \mathbf{P}_N\}$, $j \neq i$ **do**
      Add $\mathbf{P}_i \cap \mathbf{P}_j$ to $\mathbf{O}_i$
    $\overline{\mathbf{P}}_i \leftarrow \mathbf{P}_i \setminus \mathbf{O}_i$
    Add $\overline{\mathbf{P}}_i$ to $\overline{\mathcal{P}}$
  $\mathbf{E}_c \leftarrow \mathcal{M}_h(\overline{\mathcal{P}})$
  $\tilde{\mathbf{E}}_c \leftarrow \mathcal{M}_h(\overline{\mathcal{P}}, \mathbf{E}_c)$
  Get parent $\mathbf{P}$ from `Partition Tree`
  $G_{\mathbf{P}} \leftarrow G\left(\mathbf{P}, (\bigcup_i \mathbf{E}_i) \cup \tilde{\mathbf{E}}_c\right)$
  Remove nodes $\mathbf{P}_1, \ldots, \mathbf{P}_N$ from `Partition Tree`
  Return $G_{\mathbf{P}}$, `Partition Tree`

Chapter 5

Experiments

This chapter focuses on the empirical study of the method detailed in chapter 4. Given the method's modular structure, a separate evaluation of the *Divide* and *Conquer* phases is conducted, comparing the proposed strategies to baselines. Finally, section 5.3 reports the final evaluation on the CausalMan man, along with results for baselines and other methods from literature.

During experiments, the base LLM used for the system's agents was the open-source Qwen3-14B [47], while the chosen embedding model was Qwen3-Embedding-0.6B [40], both running locally on a `vLLM` server [105].

Regarding Human-in-the-Loop interactions, the users replied to agents' queries in natural language, simulating answers from a domain expert that is familiar with the dataset and its context. For narrow causal queries (e.g., "Does $X \longrightarrow Y$ hold?"), answers were derived directly from the available ground truth of the benchmark dataset. For broader, open-ended queries, responses were limited to domain knowledge, avoiding unintended hints about the ground truth.

5.1 Standalone Divide Evaluation

5.1.1 Experiment Setting

This ablation corresponds to just running the `Explain` and `Divide` module, obtaining a partitioning of the variable set $\mathbf{V}$.

The three original LLM-aided methods discussed in section 4.5, *Meta-Agents*, *Hybrid-Agents* and *Hybrid-Embeddings*, are compared to three purely "data-driven" baselines.

Method	CCE ↓	Avg Overlap ↑	Avg Size ↓	Max Size ↓
MHC	0.558 ± 0.004	0	37	71
CAPA	0.007 ± 0.001	0.78 ± 0.02	113	183
Data	0.09 ± 0.01	0.06 ± 0.04	6	174
Hybrid-Embeddings	0.76 ± 0.06	0.23 ± 0.05	12	52
Hybrid-Agents	0.90 ± 0.04	0.05 ± 0.02	4	10
Meta-Agents	0.88 ± 0.06	0.02 ± 0.01	4	9

Table 5.1: *Divide* method evaluation on the *CausalMan Medium* benchmark (N=186). Average results over 5 runs

The first is *MHC*, the Modified Hierarchical Clustering algorithm used by [74]. It uses a correlation-based distance metric $(1 - \rho_{X,Y})$ for an initial hierarchical clustering, and then finds the level in the clustering dendrogram that maximizes number of "large" (of size $\geq n/20$) clusters. On that level, smaller clusters are merged into the closest "large" clusters.

CAPA is a recreation of the partitioning strategy in CAPA [36]. It performs lower order CI tests, much like in to the `CI_CD_Partitioning` procedure described in algorithm 2, without LLM clustering. To obtain partitions, *CAPA* runs an original algorithm, rather than community detection.

Finally, *Data* is an ablation of the *Hybrid* methods, with no aid from LLMs. CI tests are done with no restriction and no agents modify the partitioning obtained from the `CI_CD_Partitioning` procedure (refer to section 4.5).

For the *Meta-Agents* and *Hybrid-Agents* method, the hyperparameter k dictating the maximum allowed partition size is set at $k = 10$.

The employed metrics are the Causal Cut Error (CCE), the average overlap ratio between sibling partitions[1], the average and maximum partition[2] size across runs.

5.1.2 Discussion

This isolated evaluation displays a trade-off between obtaining favorable metrics and producing practically useful partitions. Data-driven methods, particularly CAPA, achieve lower CCE and highest overlap scores. However, this is largely explained

[1]The two were defined in section 2.1.3

[2]Specifically, leaf partitions in the *Partition Tree*, i.e. the smallest partitions

by the tendency to form at least one disproportionately large subgroup of variables. While smaller clusters are produced as well, partitions containing almost the entire dataset (or very large subsets, such as 52 nodes in the Hybrid-Embeddings case) undermine the objective of the Divide-and-Conquer approach, which is to reduce complexity.

Agent-based methods consistently produce partitions that satisfy the initial constraint of $k = 10$. This results in groups that are more balanced and closer to the intended output of the *Divide* step. Nevertheless, these methods yield high CCE values, indicating that the resulting partitions do not fully enclose relationships among variables.

Thus, the subsequent *Combine* phase must compensate for this limitation. For both Hybrid-Agents and Meta-Agents, merging local graphs should explicitly introduce edges between partitions to preserve causal relations that may have been separated in the *Divide* stage. While this adjustment means the obtained partitioning is not "causal" in the strict sense defined in section 2.1.3, this practice is consistent with strategies adopted in prior work (see Section 3.1), and may prove to be a more effective balance between tractability and accuracy.

During the following evaluations, the chosen *Divide* method is Meta-Agents, as it yields similar group sizes and CCE as Hybrid-Agents, but takes less time to run.

5.2 Standalone Conquer Evaluation

5.2.1 Experiment Setting

To evaluate the performance of different CD methods on subgroups, it is possible to use the same metrics as the overall evaluation, namely Precision, Recall, F1 Score and Structural Hamming Distance (SHD)[3]. For each method, the average of each metric is computed on each of the leaf partitions.

The *Agents* method, relying only on metadata, and the hybrid *Agents-FCI* method were compared against the baseline FCI algorithm equipped with χ^2 independence tests, and the score-based DAGMA [16] which fits a non-linear (Multi-Layer Perceptron) model.

Precision, Recall, and SHD for each learned subgraph $G(\mathbf{P}_i, \mathbf{E}_i)$ were computed against a subset of the ground-truth graph G_{GT}. This was obtained by taking the

[3]Such metrics are defined in section 2.1.2

Method	$F_1 \uparrow$	Precision ↑	Recall ↑	SHD ↓
FCI	0	0	0	4.6 ± 4.7
DAGMA	0.1 ± 0.3	0.1 ± 0.3	0.1 ± 0.3	4.5 ± 1.8
Agents	0.86 ± 0.18	0.92 ± 0.17	0.6 ± 0.4	1.0 ± 1.1
Agents-FCI	0.3 ± 0.4	0.4 ± 0.5	0.5 ± 0.4	4.3 ± 5.3

Table 5.2: *Conquer* method evaluation on *CausalMan Medium* benchmark (N=186). Average results acroos partitions over 2 runs. Note: If SHD is 0, F_1 is set to 1, if precision and recall are both 0, F_1 is set to 0

subset of $G_{GT}(\mathbf{V}, \mathbf{E}_{GT})$ containing only the nodes in $\mathbf{P}_i \subset \mathbf{V}$ and the edges between them. No additional edges were introduced to account for excluded confounders or mediators.

5.2.2 Discussion

The results in Table 5.2 indicate a clear advantage of the *Agents* method. This outcome is expected: although the partitions (obtained with Meta-Agents) are of manageable size, they still contain a mixture of variable types (both discrete and continuous), and causal insufficiency remains present, even further exacerbated by the partitioning process.

It is also worth noting that the evaluation procedure for this standalone setting has some limitations. In particular, restricting the ground-truth graph to each partition without adjustments ignores cases where excluded variables act as confounders or mediators. A more refined construction would introduce bi-directed edges when a confounder is omitted, or directed edges when a mediator is omitted. This simplification may bias the Precision metric.

Still, FCI consistently performs the worst. Its low Recall implies inability to recover even the edges preserved in the restricted ground truth. DAGMA provides a slight improvement, likely due to reduced sensitivity to variable type, but its performance remains weak.

Agents-FCI performs worse compared to Agents alone. This can be explained by the tendency of FCI to add edges accounting for potential latent confounders or mediators, which inflates SHD and reduces precision. These "extra" edges would in principle be eliminated during the *Combine* phase [4], but within this isolated

[4] As described in section 4.7.1

evaluation they are penalized directly

Overall, *Agents* achieves both the highest precision and the lowest SHD, while also maintaining higher recall than the alternatives. For this reason, it is adopted as the *Conquer* method in the complete benchmark.

5.3 Complete Benchmark

5.3.1 Experiment Setting

Similarly for the previous standalone evaluation, F_1 and SHD scores are computed on the complete causal graph. The chosen configuration for the proposed method is as follows: the `Divide` module uses *Meta-Agents*, with hyperparameter $k = 10$, while `Conquer` uses *Agents*.

No ablation of the "Adding bridging edges" procedure within the *Combine* module[5] was done, for the reasons addressed in section 5.2.2.

Baselines include the statistical CD algorithms FCI, XGES and GranDAG, and three other LLM-based methods from literature: CausalCopilot [88], BFS [87] and PyWhyLLM [86][6].

Due to technical constraints, the LLM used within the baselines was OpenAI's o3-mini [106]. For CausalCopilot, the user let the system run, answering to its queries with all necessary information, but without adding/removing edges from its solution, and without telling it to focus on any particular group of variables. Its choice of CD algorithm was usually XGES, but during two runs where it chose the untractable FCI, the user steered it towards XGES. For PyWhyLLM, the "Augmented Suggester" version was used, which includes a RAG mechanism from the CauseNet knowledge base [107].

Both the larger *CausalMan Medium* dataset with 186 variables and a smaller 53-variable version (*CausalMan Small*) were used to benchmark the proposed method and baselines.

5.3.2 Discussion

Results in Tables 5.3 and 5.4 show consistent improvements of the proposed method over baselines.

[5] Described in section 4.7.2

[6] All of them were introduced in section 3.2

Method	F_1↑	Precision ↑	Recall ↑	SHD ↓
FCI	-	-	-	-
XGES	0.014±0	0.211±0	0.007±0	547±0
GranDAG	0±0	0±0	0±0	544±6
CausalCopilot	0.014±0	0.211±0	0.007±0	547±0
PyWhyLLM	-	-	-	-
BFS	-	-	-	-
Proposed method	0.27±0.05	0.66±0.11	0.17±0.05	558±22

Table 5.3: CD on *CausalMan Medium* benchmark ($n = 186$). A dash "-" denotes that the method failed to find a solution in a reasonable timeframe. Average results over 5 runs

Method	F_1↑	Precision ↑	Recall ↑	SHD ↓
FCI	0.028±0.012	0.021±0.005	0.04±0.06	221±3
XGES	0.056±0.009	0.037±0.006	0.12±0.10	450±15
GranDAG	0.002±0.003	0.018±0.04	0.001±0.002	113±3
CausalCopilot	0.078±0.012	0.26±0.10	0.046±0.009	117±6
PyWhyLLM	0.104±0	0.062±0	0.306±0	1,619±0
BFS	0.283±0	1±0	0.165±0	315±0
Proposed method	0.42±0.08	0.72±0.17	0.30±0.09	103±10

Table 5.4: CD on *CausalMan Small* benchmark ($n = 53$). Average results over 5 runs

On the *CausalMan Medium* dataset, several baselines failed to complete: FCI did not scale, while both PyWhyLLM and BFS exhausted the context window of o3-mini before completion. CausalCopilot ran XGES, its outputs were therefore the same as baseline XGES. Among the methods that did complete, the proposed approach yields an F_1 score an order of magnitude above baselines, maintaining balanced precision and recall. Although its SHD is not the lowest, this can be explained by the fact that some baselines produce extremely sparse graphs, which lowers SHD but also F_1 [7]. The proposed method avoids both overestimation and underestimation of edge density, obtaining a more informative causal graph.

Since scalability issues limited comparisons on the Medium benchmark, additional experiments were conducted on *CausalMan Small*. Here, the proposed method again achieves the best overall balance, with both the highest F_1 score and lowest SHD. While PyWhyLLM and BFS attain the best recall and precision respectively, they are lacking on other metrics, suggesting systematic overestimation or underestimation of causal structure.

Overall, results suggest that the proposed Divide-and-Conquer pipeline provides an effective trade-off between accuracy and scalability than existing statistical algorithms and LLM-based methods.

[7] Some points on interpretation of SHD were discussed in section 2.1.2

Chapter 6

Conclusions

This study introduced a *Divide-and-Conquer* method for causal discovery, designed to leverage state-of-the-art LLM capabilities and scale to realistic datasets with a large number of variables, mainly the simulated *CausalMan* benchmark.

The proposed method combines a multi-agent pipeline, a *Human-in-the-Loop* component and the *Divide-and-Conquer* paradigm, and thus stands as an original framework among others in causal discovery.

Standalone evaluations of the `Divide` and `Conquer` modules revealed that the LLM-based methods consistently struck a good balance between accuracy and scalability, offering practical advantages over purely data-driven baselines. In particular, the `Conquer` evaluation confirmed that LLM agents are effective at accurately reconstructing smaller causal graphs, in agreement with prior research.

Overall results are positive. The proposed agentic architecture, augmented with web information retrieval, human interaction and a memory mechanism, proved more balanced and scalable than statistical algorithms and existing LLM baselines, even on a smaller dataset of 53 variables.

However, some limitations remain. First among these is the failure to produce a partitioning that satisfies existing theory, an issue linked to an inherent trade-off between partition size and information loss during the *Divide* process. Current theoretical guarantees for *Divide-and-Conquer* algorithms are conditional on successful causal partitioning, which is not ensured in this method.

Furthermore, combining statistical and LLM-based methods did not prove effective on the *Divide* task. The most practical solution was relying solely on agents. Further refinement and experimentation are needed, especially regarding the type of

conditional independence tests employed by the proposed data-driven partitioning method: realistic datasets have mixed variable types, and an independence test that consistently handles such cases is an active topic of research.

Future improvements could also target the *Combine* phase, where the current implementation relies on heuristic connections between smaller graphs.

The empirical evaluation could be deepened with additional standalone module evaluations or ablations, as well as benchmarking against other baselines specifically designed for large-scale causal discovery.

The method itself represents a promising blueprint for scaling causal discovery with LLMs. Extending it with alternative or complementary data-driven approaches could strengthen both its performance and its theoretical guarantees.

6.1 Answers to research questions

I. LLM agents, when augmented with knowledge retrieval and human input, proved effective contributors to causal discovery, especially in reconstructing smaller causal graphs

II. LLMs were most useful in the *Conquer* and *Divide* step, however the latter remained challenging and theoretically unresolved. In the proposed *Combine* heuristic, agents successfully add relevant connections between smaller graphs

III. Current theoretical guarantees could not be fully satisfied, since the proposed partitions did not meet the strict requirements of causal partitioning

IV. Data-driven approaches and LLMs showed limited complementarity in the *Divide* phase, but LLM-only solutions offered the best practical trade-offs

Bibliography

[1] Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys, Vol. 3*, 2009.

[2] Judea Pearl. *Causality*. Cambridge university press, 2009.

[3] Otis Dudley Duncan. *Introduction to structural equation models*. Elsevier, 2014.

[4] Arthur S Goldberger. Structural equation models: An overview. *Structural equation models in the social sciences*, pages 1–18, 1973.

[5] Alessio Zanga, Elif Ozkirimli, and Fabio Stella. A survey on causal discovery: theory and practice. *International Journal of Approximate Reasoning*, 151:101–129, 2022.

[6] Judea Pearl, Madelyn Glymour, and Nicholas P Jewell. *Causal inference in statistics: A primer*. John Wiley & Sons, 2016.

[7] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[8] Ronald A Fisher. On the" probable error" of a coefficient of correlation deduced from a small sample. *Metron*, 1:3–32, 1921.

[9] Karl Pearson. *On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to have Arisen from Random Sampling*, pages 11–28. Springer New York, New York, NY, 1992.

[10] Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006.

[11] Kun Zhang, Jonas Peters, Dominik Janzing, and Bernhard Schoelkopf. Kernel-based conditional independence test and application in causal discovery, 2012.

[12] Oliver Schacht and Biwei Huang. A fast kernel-based conditional independence test with application to causal discovery, 2025.

[13] Eric V. Strobl, Kun Zhang, and Shyam Visweswaran. Approximate kernel-based conditional independence tests for fast non-parametric causal discovery. *arXiv preprint arXiv:1702.03877*, 2017.

[14] Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2000.

[15] Peter L Spirtes, Christopher Meek, and Thomas S Richardson. Causal inference in the presence of latent variables and selection bias. *arXiv preprint arXiv:1302.4983*, 2013.

[16] Kevin Bello, Bryon Aragam, and Pradeep Ravikumar. Dagma: Learning dags via m-matrices and a log-determinant acyclicity characterization, 2023.

[17] Alain Hauser and Peter Bühlmann. Characterization and greedy learning of interventional markov equivalence classes of directed acyclic graphs. *The Journal of Machine Learning Research*, 13(1):2409–2464, 2012.

[18] Amin Jaber, Jiji Zhang, and Elias Bareinboim. Causal identification under markov equivalence, 2018.

[19] Vineet K Raghu, Joseph D Ramsey, Alison Morris, Dimitrios V Manatakis, Peter Sprites, Panos K Chrysanthis, Clark Glymour, and Panayiotis V Benos. Comparison of strategies for scalable causal discovery of latent variable models from mixed data. *International journal of data science and analytics*, 6(1):33–45, 2018.

[20] Orientation rules for the PC algorithm. `https://www.rdocumentation.org/packages/MXM/versions/0.9.7/topics/Orientation%20rules%20for%20the%20PC%20algorithm`, 2016. RDocumentation page for the MXM R package (version 0.9.7).

[21] Jiji Zhang. On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. *Artificial Intelligence*, 172(16-17):1873–1896, 2008.

[22] Ayesha R. Ali, Thomas S. Richardson, Peter L. Spirtes, and Jiji Zhang. Towards characterizing markov equivalence classes for directed acyclic graphs with latent variables, 2012.

[23] Yangbo He, Jinzhu Jia, and Bin Yu. Counting and exploring sizes of markov equivalence classes of directed acyclic graphs. *The Journal of Machine Learning Research*, 16(1):2589–2609, 2015.

[24] David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

[25] Alessandro Magrini, Stefano Di Blasi, Federico Mattia Stefanini, et al. A conditional linear gaussian network to assess the impact of several agronomic settings on the quality of tuscan sangiovese grapes. *Biometrical Letters*, 54(1):25–42, 2017.

[26] Ruibo Tu, Kun Zhang, Bo Christer Bertilson, Hedvig Kjellström, and Cheng Zhang. Neuropathic pain diagnosis simulator for causal discovery algorithm evaluation, 2019.

[27] Marco Scutari, Maintainer Marco Scutari, and Hiton-PC MMPC. Package 'bnlearn'. *Bayesian network structure learning, parameter learning and inference, R package version*, 4(1), 2019.

[28] Erdogan Taskesen. Forward sampling — bnlearn documentation. `https://erdogant.github.io/bnlearn/pages/html/Sampling.html`, 2022. Accessed: 2025-09-01.

[29] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. Dags with no tears: Continuous optimization for structure learning, 2018.

[30] Ashka Shah, Adela DePavia, Nathaniel Hudson, Ian Foster, and Rick Stevens. Causal discovery over high-dimensional structured hypothesis spaces with causal graph partitioning. *Transactions on Machine Learning Research (TMLR)*, 2025.

[31] Hui Liu, Shuigeng Zhou, Wai Lam, and Jihong Guan. A new hybrid method for learning bayesian networks: Separation and reunion. *Knowledge-Based Systems*, 121:185–197, 2017.

[32] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72, 1991.

[33] Nicholas Tagliapietra, Juergen Luettin, Lavdim Halilaj, Moritz Willig, Tim Pychynski, and Kristian Kersting. Causalman: A physics-based simulator for large-scale causality, 2025.

[34] Ruichu Cai, Zhenjie Zhang, and Zhifeng Hao. Sada: A general framework to support robust causation discovery. In *International conference on machine learning*, pages 208–216. PMLR, 2013.

[35] Shuyu Dong, Michèle Sebag, Kento Uemura, Akito Fujii, Shuang Chang, Yusuke Koyanagi, and Koji Maruhashi. Dcilp: A distributed approach for large-scale causal structure learning, 2025.

[36] Hao Zhang, Shuigeng Zhou, Chuanxu Yan, Jihong Guan, Xin Wang, Ji Zhang, and Jun Huan. Learning causal structures based on divide and conquer. *IEEE Transactions on Cybernetics*, PP:1–12, 08 2020.

[37] Xiangyu Wang, Taiyu Ban, Lyuzhou Chen, Derui Lyu, Qinrui Zhu, and Huanhuan Chen. Large-scale hierarchical causal discovery via weak prior knowledge. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[38] Amita Kapoor, Antonio Gulli, Sujit Pal, and Francois Chollet. *Deep Learning with TensorFlow and Keras: Build and deploy supervised, unsupervised, deep, and reinforcement learning models.* Packt Publishing Ltd, 2022.

[39] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.

[40] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

[41] OpenAI. New embedding models and api updates. `https://openai.com/index/new-embedding-models-and-api-updates/`, January 2024. Accessed 2025-08-05.

[42] Adji B. Dieng, Francisco J. R. Ruiz, and David M. Blei. Topic modeling in embedding spaces, 2019.

[43] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A compre-

hensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72, 2025.

[44] Leonardo Berti, Flavio Giorgi, and Gjergji Kasneci. Emergent abilities in large language models: A survey, 2025.

[45] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022.

[46] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025.

[47] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

[48] Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey, 2023.

[49] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models, 2022.

[50] OpenAI. Function calling and other api updates. `https://openai.com/`

index/function-calling-and-other-api-updates/, June 2023. Blog post, announced on June 13, 2023.

[51] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan,

Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

[52] Hongshen Xu, Zihan Wang, Zichen Zhu, Lei Pan, Xingyu Chen, Lu Chen, and Kai Yu. Alignment for efficient tool calling of large language models, 2025.

[53] Cunxiang Wang, Xiaoze Liu, Yuanhao Yue, Xiangru Tang, Tianhang Zhang, Cheng Jiayang, Yunzhi Yao, Wenyang Gao, Xuming Hu, Zehan Qi, Yidong Wang, Linyi Yang, Jindong Wang, Xing Xie, Zheng Zhang, and Yue Zhang. Survey on factuality in large language models: Knowledge, retrieval and domain-specificity, 2023.

[54] Yuxia Wang, Minghan Wang, Muhammad Arslan Manzoor, Fei Liu, Georgi Georgiev, Rocktim Jyoti Das, and Preslav Nakov. Factuality of large language models: A survey, 2024.

[55] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. Language models as knowledge

bases?, 2019.

[56] Yiheng Liu, Hao He, Tianle Han, Xu Zhang, Mengyuan Liu, Jiaming Tian, Yutong Zhang, Jiaqi Wang, Xiaohui Gao, Tianyang Zhong, Yi Pan, Shaochen Xu, Zihao Wu, Zhengliang Liu, Xin Zhang, Shu Zhang, Xintao Hu, Tuo Zhang, Ning Qiang, Tianming Liu, and Bao Ge. Understanding llms: A comprehensive overview from training to inference, 2024.

[57] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021.

[58] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation, 2023.

[59] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2022.

[60] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[61] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark, 2023.

[62] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. Agentic retrieval-augmented generation: A survey on agentic rag, 2025.

[63] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[64] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024.

[65] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu,

Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[66] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[67] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

[68] Kexin Huang, Ying Jin, Ryan Li, Michael Y. Li, Emmanuel Candès, and Jure Leskovec. Automated hypothesis validation with agentic sequential falsifications, 2025.

[69] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023.

[70] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

[71] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

[72] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021.

[73] Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. Large language models for supply chain optimization, 2023.

[74] Jiaying Gu and Qing Zhou. Learning big gaussian bayesian networks: partition, estimation, and fusion, 2019.

[75] Bryon Aragam and Qing Zhou. Concave penalized estimation of sparse gaussian bayesian networks. *Journal of Machine Learning Research*, 16(69):2273–2328, 2015.

[76] Jireh Huang and Qing Zhou. Partitioned hybrid learning of bayesian network structures. *Machine Learning*, 111(5):1695–1738, 2022.

[77] Hui Ouyang, Cheng Chen, and Ke Tang. Divide-and-conquer strategy for large-scale dynamic bayesian network structure learning. In Zhongzhi Shi, Jim Torresen, and Shengxiang Yang, editors, *Intelligent Information Processing XII*, pages 63–78, Cham, 2024. Springer Nature Switzerland.

[78] Shohei Shimizu. Lingam: Non-gaussian methods for estimating causal structures. *Behaviormetrika*, 41(1):65–98, 2014.

[79] Luis M. De Campos. A scoring function for learning bayesian networks based on mutual information and conditional independence tests. *Journal of Machine Learning Research*, 7:2149 – 2187, 2006. Cited by: 273.

[80] José A Gámez, Juan L Mateo, and José M Puerta. Learning bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. *Data Mining and Knowledge Discovery*, 22(1):106–148, 2011.

[81] Xianchao Xie and Zhi Geng. A recursive method for structural learning of directed acyclic graphs. *The Journal of Machine Learning Research*, 9:459–483, 2008.

[82] Moritz Willig, Matej Zečević, Devendra Singh Dhami, and Kristian Kersting. Can foundation models talk causality? *arXiv preprint arXiv:2206.10591*, 2022.

[83] Matej Zečević, Moritz Willig, Devendra Singh Dhami, and Kristian Kersting. Causal parrots: Large language models may talk causality but are not causal. *arXiv preprint arXiv:2308.13067*, 2023.

[84] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[85] Hao Duong Le, Xin Xia, and Chen Zhang. Multi-agent causal discovery using large language models, 2024.

[86] Emre Kiciman, Robert Ness, Amit Sharma, and Chenhao Tan. Causal reasoning and large language models: Opening a new frontier for causality. *Transactions on Machine Learning Research*, 2023.

[87] Thomas Jiralerspong, Xiaoyin Chen, Yash More, Vedant Shah, and Yoshua Bengio. Efficient causal graph discovery using large language models, 2024.

[88] Xinyue Wang, Kun Zhou, Wenyi Wu, Har Simrat Singh, Fang Nan, Songyao Jin, Aryan Philip, Saloni Patnaik, Hou Zhu, Shivam Singh, Parjanya Prashant, Qian Shen, and Biwei Huang. Causal-copilot: An autonomous causal analysis agent, 2025.

[89] Masayuki Takayama, Tadahisa Okuda, Thong Pham, Tatsuyoshi Ikenoue, Shingo Fukuma, Shohei Shimizu, and Akiyoshi Sannai. Integrating large language models in causal discovery: A statistical causal approach, 2025.

[90] Wei Xu, Gang Luo, Weiyu Meng, Xiaobing Zhai, Keli Zheng, Ji Wu, Yanrong Li, Abao Xing, Junrong Li, Zhifan Li, et al. Mragent: an llm-based automated agent for causal knowledge discovery in disease via mendelian randomization. *Briefings in Bioinformatics*, 26(2):bbaf140, 2025.

[91] ChengAo Shen, Zhengzhang Chen, Dongsheng Luo, Dongkuan Xu, Haifeng Chen, and Jingchao Ni. Exploring multi-modal data with tool-augmented llm agents for precise causal discovery. *arXiv preprint arXiv:2412.13667*, 2024.

[92] Sven Meier, Pratik Narendra Raut, Felix Mahr, Nils Thielen, Jörg Franke, and Florian Risch. Structured knowledge-based causal discovery: Agentic streams of thought. *Information Processing & Management*, 62(5):104202, 2025.

[93] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems*, 135:364–381, October 2022.

[94] Vikram Singh Bisen. What is human in the loop machine learning: Why & how used in AI, May 2020. Medium, "VSINGHBISEN".

[95] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations, ICLR*, 2024.

[96] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[97] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[98] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents, 2024.

[99] LangChain AI. Langchain documentation – introduction, 2025. Accessed: 2025-09-22.

[100] LangChain AI. Langgraph documentation – overview, 2025. Accessed: 2025-09-22.

[101] Vijay Viswanathan, Kiril Gashteovski, Kiril Gashteovski, Carolin Lawrence, Tongshuang Wu, and Graham Neubig. Large language models enable few-shot clustering. *Transactions of the Association for Computational Linguistics*, 12:321–333, 04 2024.

[102] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6), December 2004.

[103] NetworkX Developers. greedy_modularity_communities — networkx 3.5 documentation. `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.modularity_max.greedy_modularity_communities.html`, 2025. Accessed: 2025-09-18.

[104] (conditional) independence tests — causal-learn documentation. `https://causal-learn.readthedocs.io/en/latest/independence_tests_index/index.html`, 2021. Causal-learn documentation, version latest.

[105] vLLM Developers. vllm documentation – getting started: Quickstart, 2025. Accessed: 2025-09-22.

[106] OpenAI. Openai o3 and o4-mini system card. `https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf`, April 2025. System card document.

[107] Stefan Heindorf, Yan Scholten, Henning Wachsmuth, Axel-Cyrille Ngonga Ngomo, and Martin Potthast. Causenet: Towards a causality graph extracted from the web. In *CIKM*. ACM, 2020.