Uncovering Hidden Correctness in LLM Causal Reasoning via Symbolic Verification

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly applied to tasks involving causal reasoning. However, current benchmarks often rely on string matching or surface-level metrics that fail to assess whether a model's output is formally valid under causal semantics. We propose DoVerifier, a symbolic verification framework that checks whether LLM-generated causal expressions are derivable from a given causal graph using rules from do-calculus and probability theory. This allows us to recover correct answers that would otherwise be marked incorrect due to superficial differences. Evaluations on synthetic data and causal QA benchmarks show that DoVerifier more accurately captures semantic correctness than standard metrics, offering a more rigorous and informative way to evaluate LLMs on causal tasks.

1 Introduction

Causal reasoning lies at the core of human intelligence. Unlike mere pattern recognition, it enables us to reason about interventions, explain effects, and predict outcomes under hypothetical scenarios. As large language models (LLMs) (OpenAI, 2024; Team, 2025; DeepSeek-AI, 2025) are increasingly deployed in scientific, medical, and policy-related domains, the ability to generate and interpret causal claims is no longer optional—it is critical (Doshi-Velez and Kim, 2017). An LLM that can distinguish between correlation and causation could support tasks ranging from experimental design to scientific hypothesis generation.

Recent causal reasoning benchmarks such as CLadder (Jin et al., 2023) and CausalBench (Wang, 2024) have begun to evaluate LLMs on causal question answering. However, these efforts primarily focus on surface-level correctness: whether the model's answer matches a gold string or produces the right outcome in simple scenarios. While useful, these metrics fail to capture a more fundamental question: *does the model's output represent a valid causal expression under formal semantics?* Furthermore, LLMs often produce expressions that are logically correct but syntactically different from the reference. These answers are penalized despite being correct, leading to an incomplete picture of model capability.

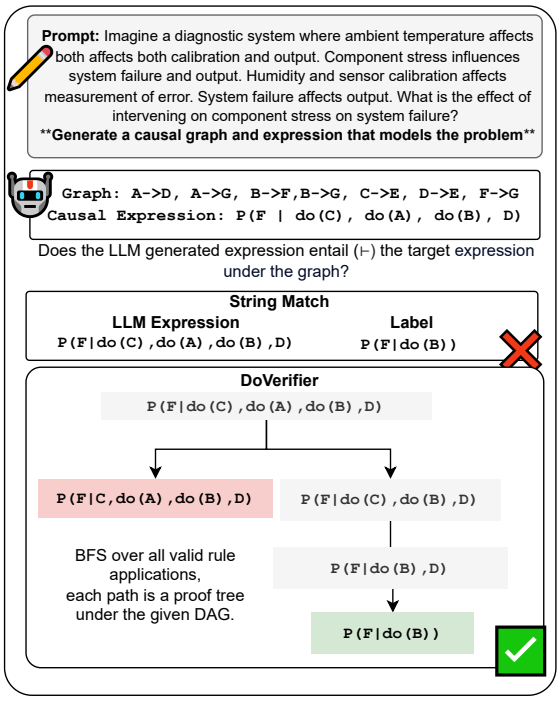

Figure 1: Our symbolic verifier checks whether a model-generated causal expression is semantically equivalent to the ground truth under a given DAG. Unlike string match, it explores all valid derivations using do-calculus and probability rules to identify formal equivalence.

This gap arises because causal inference relies on symbolic semantics: the validity of an expression like $P(Y \mid \mathrm{do}(X))$ dependents not on its string form, but on whether it is derivable from a given causal graph using rules of do-calculus and probability theory. In mathematical formalization

tasks, models can often be evaluated by plugging in values or checking numerical correctness (Gao et al., 2025; Fan et al., 2024; Cobbe et al., 2021; Hendrycks et al., 2021). However, as shown in Figure 1, we rarely know the full joint distribution $P(\cdot)$, preventing us to substitute numerical values; the ground truth is defined not by observed values, but by derivability under a causal graph using the rules of do-calculus (Pearl, 1995). This makes causal verification fundamentally symbolic: an expression like $P(Y \mid \text{do}(X))$ must be judged valid based on its formal relation to a DAG and other expressions—not via simulation or numeric output.

In this work, we propose DoVerifier, a symbolic verification framework for evaluating LLM-generated causal expressions. Given a causal graph and a model prediction, our system determines whether the expression is formally derivable using known rules. This allows us to recover many semantically correct outputs that existing benchmarks miss. As shown Figure 1, our method performs rule-based transformations to verify semantic equivalence, offering a more rigorous and informative evaluation of causal reasoning in LLMs.

We further show that symbolic verification enables strutured guidance: by identifying specific derivation failures, DoVerifier can help models revise incorrect outputs, improving causal validity without requiring gold answers.

Our contributions are as follows:

- We propose a formal verication framework for LLM-generated causal expressions, based on proof search over do-calculus and probability transformations.
- We show that DoVerifier recovers a large portion of causally correct but syntactically mismatched outputs on both synthetic data and real benchmarks, outperforming standard metrics.
- We demonstrate that symbolic verification enables feedback for model self-correction, improving causal accuracy without supervision.

2 Related Work

Causal QA and LLM Evaluation Recent benchmarks evaluate large language models (LLMs) on their ability to answer causal questions expressed in natural language. CLadder (Jin et al., 2023) and CausalBench (Wang, 2024) present standardized datasets of associational, interventional, and counterfactual queries grounded in causal graphs. However, evaluation typically hinges on string similarity to a gold-standard answer, without any guarantee of *causal validity* (Jin et al., 2023; Bondarenko et al., 2022; Joshi et al., 2024). A semantically incorrect expressions might score well due to shared tokens. Standard metrics like exact match, BLEU (Papineni et al., 2002), token-level F1, and BERTScore (Zhang et al., 2020) are commonly used to evaluate LLMs on causal QA tasks (Hu and Zhou, 2024). However, these metrics assess surface similarity, not semantic equivalence. As shown in Figure 2, they may penalize logically correct outputs due to formatting differences, or falsely reward incorrect answers that share common tokens. To our knowledge, no prior work evaluates causal QA using symbolic derivability as a criterion for semantic correctness.

Formal Verification in Causal Inference The causal inference community has long relied on do-calculus (Pearl, 1995) and probability theory to determine whether a causal query is identifiable from observational data. Classical identifiability algorithms (Shpitser and Pearl, 2008) and modern tools like `dosearch` (Tikka et al., 2021) formalize this process as a search over valid derivations. However, these tools are designed to compute causal effects from structured inputs—not to verify whether a model-generated expression is valid or equivalent under the causal graph. This verification step is critical when evaluating the intermediate reasoning of LLMs. Another line of work, like Sheth et al. (2025), checks if answers align with predefined causal graphs but relies on template matching rather than formal derivations and cannot handle expressions involving do-calculus transformations. In contrast, our approach treats verification as a symbolic proof search problem: given a model-generated expression, we check whether it can be derived from known assumptions using formally defined rules. This enables both robust evaluation and fine-grained error analysis.

Formalization in Mathematical and Logical Reasoning Efforts in mathematical reasoning have primarily focused on verifying answers to quantitative problems. For instance, Hendrycks et al. (2021) evaluates LLMs on math competition problems, while Glazer et al. (2024) investigates symbolic solvers for arithmetic tasks. To further validate intermediate reasoning steps, another line of work (Ren et al., 2025; Wang et al., 2024) resorts

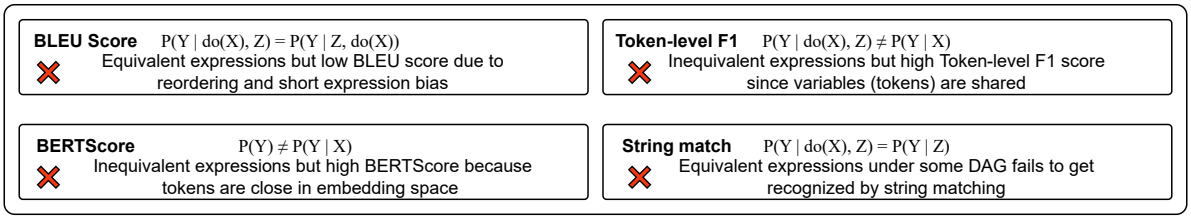

Figure 2: Examples of evaluation failures in causal expression matching. Even logically equivalent expressions can receive low scores due to surface-level differences (e.g., reordering), while inequivalent ones may score high due to shared tokens or embeddings. Highlights the limitations of BLEU, token-level F1, BERTScore, and string match in causal reasoning tasks.

to formal math descriptions (de Moura and Ullrich, 2021; Nipkow et al., 2002) that facilitate the stepwise consistency inspection. Although it is promising to *formalize* a math problem (AlphaProof and teams, 2024; Lin et al., 2025), checking its semantic correctness is found crucial yet under evolving (Lu et al., 2024; Xin et al., 2025). Recent work in geometry (Murphy et al., 2024) and logic (Li et al., 2024) uses SMT solvers to assess logical equivalence between informal text and formal theorems. We draw inspiration from this paradigm but extend it to causal inference—where correctness is defined not by logical validity alone, but by derivability under the rules of do-calculus and a causal DAG.

3 DoVerifier: Causal Symbolic Verification Framework

3.1 Preliminaries

Notation Let G be a causal DAG over a finite set of observed variables V. We define the language $\mathcal{L}_{\text{causal}}$ to consist of expressions of the form $P(Y \mid \mathbf{Z})$, where $Y \subseteq V$ and $\mathbf{Z} \subseteq V \cup \{\text{do}(X) : X \subseteq V\}$. That is, the outcome set Y includes only observed (non-interventional) variables, while the conditioning set $\mathbf{Z}$ may contain both observed variables and do-interventions.

We write $E_{\text{init}} \vdash_G E_{\text{target}}$ to denote that E_{target} is derivable from E_{init} via a finite sequence of applications of do-calculus and standard probability rules, while respecting the conditional independencies entailed by G. The graph G encodes the causal structure, which governs valid derivations and conditional independencies. We read $\vdash$ as **entails**.

Problem Statement Given a causal DAG G and two expressions $E_1, E_2 \in \mathcal{L}_{\text{causal}}$, we aim to design a model $\mathcal{F} : \mathcal{L}_{\text{causal}} \times \mathcal{L}_{\text{causal}} \times \mathcal{G} \to [0, 1]$ that determines whether E_1 and E_2 are equivalent under G. Formally, this corresponds to checking whether $E_1 \vdash_G E_2$ holds.

For example, in Figure 1, let

$$E_1 = P(F \mid \text{do}(C), \text{do}(A), \text{do}(B), D),$$
$$E_2 = P(F \mid \text{do}(B)),$$

and let G be the DAG defined by the following edges:

$$A \to D, \quad A \to G, \quad B \to F,$$
$$B \to G, \quad C \to E, \quad D \to E, \quad F \to G.$$

If $\mathcal{F}$ simply performs string matching, then $\mathcal{F}(E_1, E_2, G) = 0$. However, this is incorrect, since $E_1 \vdash_G E_2$ holds. Our goal is to design a model that captures such equivalences more accurately.

Unlike factual QA, causal evaluation is not always numeric: we cannot simply plug in values to verify an answer. Instead, we must determine whether an expression like $P(Y \mid (X))$ follows logically from a known graph structure.

To determine whether $E_1 \vdash_G E_2$ holds, we rely on the rules of do-calculus and standard probability theory (Pearl, 1995). These rules define how causal expressions in $\mathcal{L}_{\text{causal}}$ can be transformed while preserving validity under a given causal graph G. Since causal expressions may involve interventions (via $\text{do}(\cdot)$), simple syntactic matching is insufficient. Instead, entailment depends on the structure of G and the conditional independencies it encodes.

Do-calculus provides a sound and complete set of transformation rules for this purpose. In our setting, these rules form the basis for reasoning about equivalence between expressions and are central to how we define and implement $\mathcal{F}$.

We now introduce the formal rules that underpin our verification method. These rules form the core component of DoVerifier. Do-calculus consists of

three rules that specify when we can remove or add terms to a conditional distribution involving interventions (Pearl, 1995).

The Rules of *do*-calculus Let X, Y, Z, and W be arbitrary disjoint sets of nodes in a causal directed acyclic graph (DAG) G [1]. Following the notation of Pearl (2012), we denote:

- $G_{\overline{X}}$ the graph obtained from G by removing all the edges pointing to the nodes in X.
- $G_{\underline{X}}$ the graph obtained by deleting all the edges emerging from the nodes in X.
- $G_{\overline{X}\underline{Z}}$ the graph obtained by deleting edges into X and out of Z.

Each rule applies only if a certain d-seperation condition holds in the modified graph.

Rule 1 (Insertion/deletion of observations):

$$P(y \mid \mathrm{do}(x), z, w) = P(y \mid \mathrm{do}(x), w) \quad \text{if } (Y \perp\!\!\!\perp Z \mid X, W)_{G_{\overline{X}}} \tag{1}$$

This allows us to add or remove observed variables Z from the conditioning set if they are irrelevant to Y once X and W are known (after intervention X).

Rule 2 (Action/observation exchange):

$$P(y \mid \mathrm{do}(x), \mathrm{do}(z), w) = P(y \mid \mathrm{do}(x), z, w) \quad \text{if } (Y \perp\!\!\!\perp Z \mid X, W)_{G_{\overline{X}\underline{Z}}} \tag{2}$$

This allows us to replace an intervention $\mathrm{do}(Z)$ with a simple observation, if Z behaves like a non-manipulated variable under this graphical condition.

Rule 3 (Insertion/deletion of actions):

$$P(y \mid \mathrm{do}(x), \mathrm{do}(z), w) = P(y \mid \mathrm{do}(x), w) \quad \text{if } (Y \perp\!\!\!\perp Z \mid X, W)_{G_{\overline{XZ(W)}}} \tag{3}$$

This allows us to ignore an intervention on Z when it has no causal effect on Y, given the rest of the variables.

[1] In do-calculus, X, Y, Z, and W are disjoint sets of variables representing interventions (X), outcomes (Y), observed variables (Z), and other variables (W). These sets can be empty which allows the rules to generalize to many causal inference scenarios.

Notation: $Z(W)$ is the set of Z-nodes that are *not* ancestors of any W-node in $G_{\overline{X}}$. This restrictions ensures we only remove do-interventions that don't "leak" back into relevant parts of the graph. The notation $(Y \perp\!\!\!\perp Z \mid X, W)_G$ represents d-separation in graph G, meaning all paths between Y and Z are blocked by conditioning on X and W.

$$A \to D, \quad A \to G, \quad B \to F, \quad B \to G, \quad C \to E, \quad D \to E, \quad F \to G.$$

3.2 Method

We define a symbolic verification framework, DoVerifier, for assessing equivalence between causal expressions derived from natural language. Later in the paper we discuss why existing metrics fail to meet such desired properties. Given a causal DAG G (which may be generated by the model), and two expressions $E_{\text{init}}, E_{\text{target}} \in \mathcal{L}_{\text{causal}}$, DoVerifier determines whether E_{target} is derivable from E_{init} under the axioms of do-calculus and standard probability theory. The system operates by enumerating proof sequences through a structured search procedure. Implementation details are provided in Appendix B. In simple terms, our framework is simply breadth-first-search after converting everything into a custom object. The framework consists of two main components:

1. **Expression Parser.** This module parses expressions from natural language or symbolic form into normalized structured representations in $\mathcal{L}_{\text{causal}}$. This includes:
   - Recognizing both observational terms like $P(Y \mid X)$ and interventional ones like $P(Y \mid \mathrm{do}(X), Z)$.
   - Converting string-based expressions into a canonical symbolic form using a custom SymPy-based object[2] that allows equivalence checks that are invariant to variable reodering or formatting.
   - If a causal graph is provided, it is parsed into a standard NetworkX [3] DAG object.

   This step is necessary to interface LLM outputs with our proof search module.

2. **Proof Search Module.** This module determines whether a valid derivation exists from E_{init} to E_{target} under the rules of do-calculus and probability theory.

[2] https://www.sympy.org
[3] https://networkx.org/

- We implement a breadth-first search over the space of derivable expressions. The algorithm is provided in Appendix B.
- At each step, we apply all possible transformations to current expressions and enqueue any new expressions not seen before.
- The search continues until the target expression is found or a predefined depth limit is reached (e.g., 20 steps).

This guarantees completeness within a bounded search space: if a derivation exists within that limit, it will be found.

3.3 Soundness and Completeness of DoVerifier

In this section, we establish a theoretical guarantee for DoVerifier by proving its soundness and completeness. We begin by introducing formal notations that support these proofs. Soundness ensures that every equivalence established by DoVerifier is valid; completeness ensures that if an equivalence exists between two expressions, DoVerifier will find it (within a bounded search depth). First, we model causal equivalence as a reachability problem in a derivation graph:

Proposition 3.1 (Derivation Graph)**.** *Let $E_{init} \in \mathcal{L}_{causal}$. Define a directed graph $S(E_{init})$ where:*

- *Each node is a unique causal expression derivable from E_{init};*
- *An edge $E \to E'$ exists if E' can be obtained from E by applying a single valid transformation.*

Then $S(E_{init})$ is a well-defined, finite-branching graph.

The branching factor is finite since the number of variables in G is finite and each transformation rule applies to bounded subsets.

Proof. Proved in Appendix C ■

Verification Algorithm Given a causal graph G, source expression E_{init}, target expression E_{target}, and maximum depth d, we present Algorithm 1 as an algorithm to verify if E_{init} and E_{target} are equivalent bounded by depth d.

This approach guarantees finding the shortest sequence of transformations if one exists within the depth limit, as stated in our main theorem that concerns the soundness and completeness of the verification algorithm:

Proposition 3.2 (Soundness & Completeness of Proof Search)**.** *Let G be a causal DAG, and let $E_{init}, E_{target} \in \mathcal{L}_{causal}$. If $E_{init} \vdash_G E_{target}$, then Algorithm 1 returns a valid proof sequence within depth d, for some finite d. Conversely, if no such derivation exists within depth d, Algorithm 1 returns* None.

Proof. Proved in Appendix D. ■

In addition, if no derivation exists between E_{init} and E_{target} with $k \leq d$ steps, breadth-first search (BFS) will terminate after exploring all expressions within depth d. Further practical considerations are explained in Appendix E

Separately, we can view DoVerifier as a logical system defined over a formal language $\mathcal{L}_{\text{causal}}$ equipped with a set of derivation rules $\mathcal{R}$ and a background model that is either provided or generated by an LLM G (the DAG). In this view, we distinguish between:

Syntactic entailment ($H \vdash_G A$): A is derivable from hypothesis H using the symbolic transformation rules admissible under G bounded by depth d.

Semantic entailment ($H \models_G A$): A is true in all causal models consistent with G in which all $H_i \in H$ are true.

Intuitively, **syntactic entailment** means that there exists a proof of A from H using valid rules (i.e., it can be derived step by step). In contrast, **semantic entailment** means that A holds in every causal model where H is true, regardless of how we prove it.

An analogy: syntactic entailment is like showing your work on a math problem using allowed steps; semantic entailment is like checking that the answer is always correct in every valid scenario. In our setting, the two notions coincide because do-calculus is sound and complete—everything provable is true, and everything true is provable. We require our inference system to satisfy:

Soundness: If $H \vdash_G A$, then $H \models_G A$

Completeness: If $H \models_G A$, then $H \vdash_G A$

These are properties of the underlying logical system, which are satisfied due to the completeness of

do-calculus for causal identifiability (Pearl, 1995) and the standard probability axioms. Thus, the semantic correctness of model outputs can be equivalently verified through syntactic derivation, which forms the basis of our verifier.

4 Experiments and Results

We evaluate our verifier in two complementary settings.

Synthetic Data. This setting enables precise control over the graph structure, derivation path, and rule usage. It allows us to test whether the verifier behaves correctly under known conditions, and whether it can recover known valid derivations.

LLM Causal Reasoning. This setting evaluates the verifier's practical utility in assessing causal reasoning in free-form LLM outputs. Here, we test whether the verifier can recognize semantically correct answers that would be missed by surface-level metrics such as string match.

4.1 Synthetic Data Test

To verify the internal consistency of the verifier, and to show that existing metrics fail in cases where syntactically different expressions are the same semantically, we construct a synthetic dataset of over 10,000 expression pairs $(E_{\text{init}}, E_{\text{target}})$ such that E_{target} is provably derivable from E_{init} under a known DAG G. Each pair involves between 1–4 rule applications and includes randomized use of do-calculus and probability rules $\mathcal{P}$. A description of data samples is shown in Appendix F.

Sampling Procedure Let $V = \{v_1, \ldots, v_n\}$ be a finite set of variables, and let $G = (V, E)$ be a randomly sampled acyclic graph. We sample the directed edges independently as $\mathbb{P}(v_i \to v_j) = p$ for $i < j$ where $p \in (0, 1)$ is the edge probability, and the ordering ensures the graph is acyclic. In our experiments, we fix $n \leq 10$ and $p = 0.5$ to balance expressivity and tractability. We first construct

$$e_1 = P(Y \mid \text{do}(X_1), \ldots, \text{do}(X_k), Z_1, \ldots, Z_m) \tag{4}$$

where $Y \in V$ is chosen uniformly at random, a subset of $V \backslash \{Y\}$ is chosen as intervention variables $\{X_i\}$ and additional variables $\{Z_j\}$ are included as conditioning set as observation. To ensure structural diversity, the number of intervention variables X_i and observational variables Y_i is randomly chosen per sample, subject to DAG constraints. Then, we define a symbolic derivation process π consisting of a sequence of rule applications:

$$e_1 \xrightarrow{r_1} e_2 \xrightarrow{r_2} \ldots \xrightarrow{r_n} e_{n+1} \tag{5}$$

where each $r_i \in \{\text{Rule 1}, \text{Rule 2}, \text{Rule 3}\} \cup \mathcal{P}$. Rule applications are randomized but constrained to only apply when valid under the conditional independencies induced by G. Then, we set $E_{\text{init}} = e_1$ and $E_{\text{target}} = e_{n+1}$.

The mean number of edges is 7 (min. 3, max. 10). Rule 1 was used 21172 times, rule 2 was used 29563 times, and rule 3 was used 22508 times.

Synthetic Data Performance Our symbolic verifier achieves 100% precision and recall under depth limit $d = 5$, demonstrating correctness of the derivation engine, while other methods such as string match, or token-level F1 performed poorly due to E_{init} and E_{target} being too distinct syntactically.

The experiment results show a key strength of our framework that it can correctly recognize when two expressions are equivalent under the rules of do-calculus and probability, even if they differ in formatting, variable order, or surface form.

4.2 LLM Causal Reasoning Test

Uncovering Missed Correct Answers We evaluate the ability of our symbolic verifier to improve the accuracy of large language model (LLM) evaluation in causal reasoning. Specifically, we ask: *Can our method recover correct answers that are missed by naive evaluation metrics?*

Evaluated Dataset and Models To investigate this, we use the **CLadder** benchmark (Jin et al., 2023), a suite of causal questions grounded in known DAGs. Each question is paired with a ground-truth answer expressed as a formal causal expression. We prompt `Llama-3-8B` (Grattafiori et al., 2024), `Llama-3-8BInstruct` (Grattafiori et al., 2024), `Mistral-7B` (Jiang et al., 2023), and `Gemma-7B-IT` (Team et al., 2024) to answer these questions and parse their responses, including a DAG that models the problem into symbolic expressions. Detailed prompts and parsing are demonstrated in Appendix G. Each prediction is then compared to the ground-truth using three metrics:

- **String Match:** A response is marked correct only if it matches the ground-truth expression exactly (after normalizing).

Model	String Match	LLM-as-a-judge	DoVerifier (Ours)
Llama3.1-8B	0.57	0.60	**0.73**
Mistral-7B	0.58	0.80	**0.94**
Llama3.1-8B-Instruct	0.88	0.66	**0.90**
Gemma-7B-it	0.80	0.58	**0.84**

Table 1: DoVerifier recovers more correct causal expressions than string match or using an LLM-as-a-judge across four LLMs on CLadder in terms of answer accuracy; our method identifies semantically valid expressions missed by surface-level metrics.

- **LLM-as-a-judge:** We provide the generated causal expression, generated causal graph, and the ground truth expression for OpenAI's GPT-4o (OpenAI, 2024) to determine if the two expressions are equivalent.

- **Symbolic (Ours):** A response is considered correct if it is derivable from the ground-truth using valid applications of do-calculus and probability rules (under 20 steps).

Results As shown in Table 1, our symbolic method identifies more correct answers than string match and LLM-as-a-judge, raising the accuracy across all models. Our method is more useful when models such as `Llama3.1-8b` and `Mistral-7B` output an alternative form of the correct use. This improvement highlights an important phenomenon: many model responses are *causally correct* but fail naive evaluation due to superficial differences in formatting, variable order, or phrasing. The running time of verifying through BFS is minimal (milliseconds).

Our symbolic verifier recovers this missing accuracy by judging expressions based on their semantic content, not their surface form. It enables a more faithful and rigorous assessment of causal reasoning in LLMs, ensuring that models receive credit for valid reasoning even when their output does not match the reference verbatim.

When high-performing models like `Llama3.1-Instruct` already align well with ground-truth formats, so the relative gain over string match is smaller. This suggests that the benefit of symbolic evaluation is most pronounced when models exhibit partial causal understanding but struggle with precise formalization. Furthermore, when using LLM-as-a-judge, we lose the soundness guarantee that both string match and DoVerifier provide[4]. We also hypothesize that including the DAG in LLM-as-a-judge prompts may act as a source of noise, leading the model to overinterpret structural cues and misjudge otherwise valid expressions.

We identified several common patterns where symbolic verification offers substantial advantages:

Intervention with conditioning: Our system validates equivalence between expressions like $P(Y \mid \mathrm{do}(X), Z = z)$ and $P(Y \mid \mathrm{do}(X), Z)$ by correctly handling instantiated versus symbolic values.

Rule-based transformations: Our system correctly identifies that $P(Y \mid \mathrm{do}(X), Z)$ can be transformed into $P(Y \mid X, Z)$ in DAGs where Z d-separates Y from incoming edges of X. This conversion from interventional to observational queries represented the majority of all verified equivalences. Note that this is important since the ground-truth of CLadder is in observational queries.

Multi-step proofs: For more complex cases, our verifier successfully applied sequential rules.

4.3 Improving LLMs with Symbolic Feedback

Beyond evaluation, the proposed DoVerifier enables structured feedback to guide LLMs toward correct causal reasoning without relying on ground truth expressions. It has been shown that symbolic feedback loops (e.g., using SMT solvers in math or logic) have been shown to improve LLM output accuracy by providing formal, structured corrections (Hong et al., 2025; Murphy et al., 2024). Instead of using a reference answer as an oracle, our system leverages the causal graph structure and independence relationships to provide principled guidance. We provide a formal description in Appendix H.

Results Table 2 shows the improvement of LLM performance using our feedback loop. We find that the effectiveness of symbolic feedback depends

[4] While string match is sound, it is not complete.

Model	Before Feedback	After Feedback
LLaMA3.1-8B	0.73	**0.93**
Mistral-7B	0.94	**0.99**
LLaMA3.1-8B-Instruct	0.90	**0.98**
Gemma-7B-it	0.84	**0.87**

Table 2: Accuracy before and after applying verifier-guided feedback. Feedback improves semantic correctness across all models.

heavily on the type of error in the original expression. For example, when the model incorrectly uses $P(Y \mid X)$ instead of $P(Y \mid \mathrm{do}(X))$, feedback guided by d-separation and rule-based reasoning often corrects the mistake. In contrast, if the model hallucinates an irrelevant variable or misrepresents the structure of the DAG itself, our framework is less effective since the symbolic transformations cannot fix structurally flawed inputs.

5 Discussions

This work formalizes the task of verifying causal correctness in language model outputs as a symbolic inference problem. The primary objective of the study is the derivation graph $S(E_{\text{init}})$ induced by the application of a finite rule set $\mathcal{R}$ (comprising do-calculus and probability transformations) to an initial causal expressions.

Semantic Equivalence as Proof-Theoretic Reachability We define semantic equivalence with respect to a causal graph G as the symmetric closure of the derivability relation:

$$E_1 \equiv_G E_2 \iff (E_1 \vdash_G E_2 \wedge E_2 \vdash_G E_1) \quad (6)$$

This defines a family of equivalence classes $[E]_{\equiv_G} \subset \mathcal{L}_{\text{causal}}$, where each class represents all expressions that are equivalent iff they encode the same interventional distribution in all causal models consistent with G. Empirically, we observe that LLM-generated outputs frequently fall into these equivalence classes without being string-identical to reference answers. For instance, expressions like $P(Y \mid X, Z)$ and $P(Y \mid \mathrm{do}(X), Z)$ are lexically distinct but often semantically equivalent, conditional on specific d-separation statements. Our symbolic verifier resolves this not via heuristics, but by computing membership in the equivalence class through derivation.

Symbolic Feedback Works Because of Local Equivalence Neighborhoods In the presence of an incorrect LLM output E_{LLM}, our framework enables symbolic feedback by computing a correction E' such that

$$E' \in \text{Closure}_{\mathcal{R}}(E_{\text{LLM}}) \cap [E^*]_{\equiv_G} \quad (7)$$

where E^* is the latent correct expression (not known to the verifier). Operationally, this amounts to inverse proof search: finding a path from E_{LLM} to a semantically correct neighbor. This supports the hypothesis that modern LLMs operate near locally correct regions of $\mathcal{L}_{\text{causal}}$, but lack explicit guarantees of logical closure (Wei et al., 2023; Zhou et al., 2023).

Failure Types Align with Non-derivability The most common model failures (e.g., using $P(Y \mid X)$ when X is a collider, or omitting keyfounders) correspond to derivations that fail d-separation conditions. For instance, symbolic proof fails when:

$$(Y \not\vdash Z \mid X)_{G_{\overline{X}}} \implies P(Y \mid X, Z) \not\equiv_G P(Y \mid \mathrm{do}(X), Z) \quad (8)$$

These cases, which account for a significant portion of the errors in the models, are not just empirically incorrect but provably invalid under our formal system. This illustrates how symbolic reasoning captures not only surface alignment but deep structural correctness.

6 Conclusion

We introduced DoVerifier, a formal verification framework that evaluates the causal validity of LLM-generated expressions by modeling causal reasoning as a symbolic derivation task using do-calculus and probability rules. Our approach recovers semantically correct answers that are missed by standard metrics, improves recall on causal benchmarks, and enables structured feedback to refine model outputs.

These findings reveal a significant gap in current evaluation methods and highlight the importance of symbolic verification for building reliable causal reasoning systems. By connecting natural language generation with formal inference, DoVerifier offers a principled step toward evaluating models based on what they truly understand rather than how they phrase it.

Limitations

While promising, our approach has several limitations and opens up for future work. On the

one hand, the space of valid derivations can grow rapidly with the number of variables and the depth of allowed transformations. Although we employ optimizations like expression normalization and memoization, our breadth-first search remains computationally expensive in dense or deep DAGs. Future work could explore neural-guided proof search or approximate symbolic methods. On the other hand, regarding the feedback mechanism, the current feedback module improves the causal validity of model outputs using only the predicted DAG and the initial expression. It does not incorporate the original natural language question. As a result, the revised expression may be causally correct under the graph, but not necessarily faithful to the question intent. In practice, we observe that most LLM errors stem from misapplying causal semantics rather than misreading the question, but integrating question-aware feedback remains a valuable direction for future work.

Ethical Considerations

This work focuses on the formal verification of causal expressions generated by large language models (LLMs), with the goal of improving their semantic correctness and reliability in reasoning tasks. Our proposed framework does not involve human subject data, personally identifiable information, or real-world deployment in high-stakes settings such as healthcare or public policy. However, we acknowledge that causal claims can influence decision-making in sensitive domains. As such, we emphasize that symbolic correctness under do-calculus does not guarantee practical validity unless the underlying causal graph is itself accurate and contextually appropriate.

Our framework is designed for evaluation and diagnostic purposes, not for automating causal decisions. We caution against interpreting verified expressions as endorsements of correctness in real-world applications without domain expertise. To avoid misuse, we release our tools with clear disclaimers that they are intended for research and educational purposes.

References

AlphaProof and AlphaGeometry teams. 2024. Ai achieves silver-medal standard solving international mathematical olympiad problems. *https://deepmind.google/discover/blog/ ai-solves-imo-problems-at-silver-medal-level/*.

Alexander Bondarenko, Magdalena Wolska, Stefan Heindorf, Lukas Blübaum, Axel-Cyrille Ngonga Ngomo, Benno Stein, Pavel Braslavski, Matthias Hagen, and Martin Potthast. 2022. CausalQA: A benchmark for causal question answering. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3296–3308, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

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. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Leonardo de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *Preprint*, arXiv:1702.08608.

Jingxuan Fan, Sarah Martinson, Erik Y. Wang, Kaylie Hausknecht, Jonah Brenner, Danxian Liu, Nianli Peng, Corey Wang, and Michael Brenner. 2024. HARDMATH: A benchmark dataset for challenging problems in applied mathematics. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2025. Omni-MATH: A universal olympiad level mathematic benchmark for large language models. In *The Thirteenth International Conference on Learning Representations*.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, Natalie Stewart, Bogdan Grechuk, Tetiana Grechuk, Shreepranav Varma Enugandla, and Mark Wildon. 2024. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *Preprint*, arXiv:2411.04872.

Aaron Grattafiori et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Preprint*, arXiv:2103.03874.

Sungee Hong, Zhengling Qi, and Raymond K. W. Wong. 2025. Distributional off-policy evaluation with bellman residual minimization. *Preprint*, arXiv:2402.01900.

Taojun Hu and Xiao-Hua Zhou. 2024. Unveiling llm evaluation focused on metrics: Challenges and solutions. *Preprint*, arXiv:2404.09135.

Albert Q. Jiang et al. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Zhijing Jin, Yuen Chen, Felix Leeb, Luigi Gresele, Ojasv Kamal, Zhiheng LYU, Kevin Blin, Fernando Gonzalez Adauto, Max Kleiman-Weiner, Mrinmaya Sachan, and Bernhard Schölkopf. 2023. Cladder: Assessing causal reasoning in language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 31038–31065. Curran Associates, Inc.

Nitish Joshi, Abulhair Saparov, Yixin Wang, and He He. 2024. LLMs are prone to fallacies in causal inference. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10553–10569, Miami, Florida, USA. Association for Computational Linguistics.

Zenan Li, Yifan Wu, Zhaoyu Li, Xinming Wei, Xian Zhang, Fan Yang, and Xiaoxing Ma. 2024. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. *Preprint*, arXiv:2410.20936.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. 2025. Goedel-prover: A frontier model for open-source automated theorem proving. *CoRR*, abs/2502.07640.

Jianqiao Lu, Yingjia Wan, Yinya Huang, Jing Xiong, Zhengying Liu, and Zhijiang Guo. 2024. Formalalign: Automated alignment evaluation for autoformalization. *CoRR*, abs/2410.10135.

Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. 2024. Autoformalizing euclidean geometry. *Preprint*, arXiv:2405.17216.

Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer.

OpenAI. 2024. Hello gpt-4o. Accessed: 2025-04-30.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Judea Pearl. 1995. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688.

Judea Pearl. 2012. The do-calculus revisited. *Preprint*, arXiv:1210.4852.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. 2025. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*.

Ivaxi Sheth, Bahare Fatemi, and Mario Fritz. 2025. CausalGraph2LLM: Evaluating LLMs for causal queries. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2076–2098, Albuquerque, New Mexico. Association for Computational Linguistics.

Ilya Shpitser and Judea Pearl. 2008. Complete identification methods for the causal hierarchy. *Journal of Machine Learning Research*, 9(64):1941–1979.

Gemma Team. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Gemma Team et al. 2024. Gemma: Open models based on gemini research and technology. *Preprint*, arXiv:2403.08295.

Santtu Tikka, Antti Hyttinen, and Juha Karvanen. 2021. Causal effect identification from multiple incomplete data sources: A general search-based approach. *Journal of Statistical Software*, 99(5):1–40.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. 2024. LEGO-prover: Neural theorem proving with growing libraries. In *The Twelfth International Conference on Learning Representations*.

Zeyu Wang. 2024. Causalbench: A comprehensive benchmark for evaluating causal reasoning capabilities of large language models. In *Causality and Large Models @NeurIPS 2024*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Huajian Xin, Luming Li, Xiaoran Jin, Jacques Fleuriot, and Wenda Li. 2025. Ape-bench i: Towards file-level automated proof engineering of formal math libraries. *arXiv preprint arXiv:2504.19110*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. *Preprint*, arXiv:1904.09675.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. *Preprint*, arXiv:2205.10625.

A Desired Properties of a Good Verifier

A central question in the design of verifiers for symbolic causal reasoning is: what kinds of differences between derivations should not affect the evaluation? In other words, what transformations should a good evaluator be invariant to. In this section, we formalize the invariance and sensitivity properties that an ideal evaluator should satisfy. These properties are motivated both by formal semantics and by practical considerations in modeling causal reasoning.

Given an initial expression ϕ_0, a target expression ϕ^*, and a derivation sequence $\mathcal{D} = (\phi_0, \phi_1, \ldots, \phi_k = \phi^*)$, the evaluator should assign a score $s(\mathcal{D}) \in \mathbb{R}$ that reflects the logical correctness, minimality, and interpretability of the derivation.

Definition (Syntactic Equivalence). Let ϕ and ϕ' be probability expressions. We write $\phi \equiv_{\text{syn}} \phi'$ if they differ only by a syntactic permutation that preserves semantic content, such as reordering terms in a conditioning set:

$$P(Y \mid X, Z) \equiv_{\text{syn}} P(Y \mid Z, X) \tag{9}$$

Desideratum 1 (Syntactic Invariance). Let $\mathcal{D}$ be a derivation and $\mathcal{D}'$ a derivation obtained by a sequence of syntactic equivalences to the intermediate steps. Then:

$$s(\mathcal{D}) = s(\mathcal{D}') \tag{10}$$

Definition (α-Renaming). Let ϕ contain a variable V that does not appear free in other parts of the expression. Let ϕ' be the result of replacing V by V', where V' is a fresh variable name. Then $\phi \equiv_\alpha \phi'$.

Desideratum 2 (α-Renaming Invariance). The evaluator must satisfy

$$s(\mathcal{D}) = s(\mathcal{D}') \quad \text{if each } \phi'_i \equiv_\alpha \phi_i \text{ for all } i \tag{11}$$

Definition (Well-Typed Step). A step $\phi_i \rightarrow \phi_{i+1}$ using do-calculus Rule $r \in \{\text{Rule1}, \text{Rule2}, \text{Rule3}\}$ is valid if an only if the required graphical conditional independence is entailed by DAG G associated with the problem.

Desideratum 3 (Rule Sensitivity). If $\mathcal{D}$ and $\mathcal{D}'$ differ only in that $\mathcal{D}'$ includes a rule application r that violates the required independence, then:

$$s(\mathcal{D}') < s(\mathcal{D}) \tag{12}$$

This ensures the evaluator penalizes logically invalid or unsound reasoning.

Definition (Commutativity of Independent Steps). Let $\phi_i \rightarrow \phi_{i+1} \rightarrow \phi_{i+2}$ be two derivation steps, each applying a rule to a disjoint subformula of the expression. If $\mathcal{D}_1$ and $\mathcal{D}_2$ are derivations that only differ in the order of these two steps, then they are commutative.

Desideratum 4 (Step Order Invariance). We want $s(\mathcal{D}_1) = s(\mathcal{D}_2)$ if $\mathcal{D}_1, \mathcal{D}_2$ are commutative of independent steps to ensure the evaluator does not privilege arbitrary ordering of logically independent rule applications.

Definition (Derivational Equivalence). Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be distinct derivations from ϕ_0 to ϕ^*, where each step in both sequences is valid, though possibly differing in the choice or order of applied rules.

Desideratum 5 (Robustness to Valid Alternatives). The evaluator should satisfy $\forall \varepsilon > 0$:

$$|s(\mathcal{D}_1) - s(\mathcal{D}_2)| \leq \varepsilon \tag{13}$$

This encourages diversity in valid derivations without heavily penalizing alternative but correct reasoning paths.

B Implementation Details of DoVerifier

Algorithm 1 Causal Expression Equivalence Verification

```
Initialize queue Q ← [(E_init, [])]                ▷ (expression, proof path π)
Initialize visited set V ← {E_init}
while Q not empty do
    (E, π) ← Q.dequeue()
    if E = E_target then
        return π                                   ▷ Found equivalence
    end if
    if | π |< d then
        for each applicable rule r do
            E' ← apply(r, E)
            if E' ∉ V then
                V.add(E')
                Q.enqueue((E', π + [r]))
            end if
        end for
    end if
end while
return None                                        ▷ No equivalence found within depth d
```

Our implementation converts abstract causal expressions into concrete computational objects that can be manipulated through rule applications. The core components are implemented as follows:

Expression Representation We represent causal expressions using a symbolic framework built on SymPy. Each causal probability expression $P(Y \mid \text{do}(X), Z)$ is represented as a `CausalProbability` object with an outcome variable and a list of conditioning factors, which may include both observational variables and interventional variables (wrapped in `Do` objects). This representation allows for:

- Unique identification of expressions through consistent string conversion
- Distinction between interventional and observational variables
- Manipulation of expressions through rule applications

Causal Graph Representation Causal graphs are represented using NetworkX directed graphs, where nodes correspond to variables and edges represent causal relationships. For each rule application, we create modified graph structures according to the do-calculus definitions:

- For Rule 1, we remove incoming edges to intervention variables using $G_{\overline{X}}$
- For Rule 2, we remove both incoming edges to primary interventions and outgoing edges from secondary interventions using $G_{\overline{X}\underline{Z}}$
- For Rule 3, we perform the appropriate graph modifications for $G_{\overline{XZ(W)}}$ as specified by Pearl

D-separation Testing To determine rule applicability, we implement d-separation tests using NetworkX's built-in `is_d_separator` function. For each potential rule application, we:

1. Create the appropriate modified graph based on the rule
2. Identify the variables that need to be tested for conditional independence
3. Perform the d-separation test with the appropriate conditioning set
4. Apply the rule only if the independence condition is satisfied

For example, when applying Rule 1 to remove an observation Z from $P(Y \mid \text{do}(X), Z)$, we test whether Y and Z are d-separated given X in the graph $G_{\overline{X}}$.

Search Algorithm Optimization To make the breadth-first search efficient, we implement several optimizations:

- **Expression normalization:** We convert expressions to canonical string representations with consistent ordering and whitespace removal.
- **Memoization:** We cache the results of d-separation tests to avoid redundant graph operations.
- **Early termination:** We immediately return a proof path when the target expression is found.
- **Visited set tracking:** We maintain a set of already-visited expressions to avoid cycles and redundant exploration.

Handling Incomplete Knowledge A key innovation in our implementation is the ability to work with incomplete causal knowledge. When the full DAG structure is unknown, our system can:

- Work with explicitly provided independence pairs between variables
- Infer independence relationships from partial graph information
- Explore potential equivalences under different assumptions

Scope of Verification While our implementation includes representations for both probability distributions (P) and expectations (E), our current verification framework focuses on causal expressions involving probabilities. This focus aligns with Pearl's do-calculus, which was formulated for probability distributions. The identification of causal effects fundamentally involves transforming interventional probabilities into expressions based on observed data.

The framework can be extended to handle expectations directly, as we have implemented the

necessary data structures and fundamental operations for expectation expressions. However, since expectations are functionals of probability distributions, verifying equivalence at the probability level is sufficient for most practical causal inference tasks. Once the correct probability expression is identified, expectations and other functionals can be derived through standard statistical methods.

C Proof of Theorem 3.1

We restate the proposition for easier reference:

Proposition C.1 (Derivation Graph)**.** *Let $E_{init} \in \mathcal{L}_{causal}$. Define a directed graph $S(E_{init})$ where:*

- *Each node is a unique causal expression derivable from E_{init};*
- *An edge $E \to E'$ exists if E' can be obtained from E by applying a single valid transformation.*

Then $S(E_{init})$ is a well-defined, finite-branching graph.

Proof. Let G be a causal DAG with finite node set V. Let $\mathcal{L}_{\text{causal}}$ denote the set of well-formed causal expressions over V, where each expression is of the form $P(Y \mid \mathbf{Z})$ with $Y \subseteq V$ and $\mathbf{Z}$ containing observed or interventional variables (i.e., elements of V or $\text{do}(V)$). Because V is finite, so is the set of possible subsets and intervention/observation combinations, hence $\mathcal{L}_{\text{causal}}$ is countable.

Let $\mathcal{R}$ be the set of valid transformation rules (e.g., the three rules of do-calculus and standard rules of probability). Each rule $r \in \mathcal{R}$ is modeled as a partial function:

$$r : \mathcal{L}_{\text{causal}} \to \mathcal{L}_{\text{causal}}, \tag{14}$$

where $r(E)$ is defined if the syntactic and graphical preconditions (e.g., d-separation in G) for applying r to E are satisfied.

Define the **derivation relation** $\Rightarrow$ on $\mathcal{L}_{\text{causal}}$ by:

$$E \Rightarrow E' \iff \exists r \in \mathcal{R} \text{ such that } r(E) = E'.$$

We now define the derivation graph $S(E_{\text{init}})$ as a directed graph $(\mathcal{V}, \mathcal{E})$, where:

- $\mathcal{V}$ is the set of expressions reachable from E_{init} via a finite sequence of $\Rightarrow$ steps (i.e., derivable expressions);
- $\mathcal{E}$ contains an edge (E, E') if $E \Rightarrow E'$.

To prove the theorem, we must show two things:

(1) Well-definedness. The graph $S(E_{\text{init}})$ is well-defined because:

- Each expression in $\mathcal{L}_{\text{causal}}$ has a canonical syntactic representation.
- Each rule $r \in \mathcal{R}$ is a well-defined partial function whose domain is determined by decidable conditions (syntactic and graphical).
- The derivation relation $\Rightarrow$ is therefore well-defined and finitely composable.

(2) Finite branching. For any node $E \in \mathcal{V}$:

- The number of rule applications is finite, because:
  - The number of rules in $\mathcal{R}$ is finite.
  - Each rule r examines a finite number of subsets of V (e.g., X, Y, Z, W), which are at most $2^{|V|}$ in number.
  - Rules act on bounded-size fragments of expressions and generate outputs in $\mathcal{L}_{\text{causal}}$, which is countable.
- Thus, from any E, only finitely many E' satisfy $E \Rightarrow E'$, i.e., $\text{OutDegree}(E)$ is finite.

Hence, $S(E_{\text{init}})$ is a well-defined, finite-branching directed graph. ■

D Proof of Theorem 3.2

We formally prove the soundness and completeness of our verification framework by modeling it as a symbolic derivation system over a finite-branching graph induced by transformation rules.

We restate the proposition for easier reference:

Proposition D.1 (Soundness & Completeness of Proof Search)**.** *Let G be a causal DAG, and let $E_{init}, E_{target} \in \mathcal{L}_{causal}$. If $E_{init} \vdash_G E_{target}$, then Algorithm 1 returns a valid proof sequence within depth d, for some finite d. Conversely, if no such derivation exists within depth d, Algorithm 1 returns* None.

First we show that DoVerifier is sound. Suppose we are trying to find a proof sequence starting from E_{init} to E_{target}.

Proof. Assume for contradiction that DoVerifier is not sound. Then there exists some proof path $\pi = \langle E_1, E_2, \ldots, E_k \rangle$ returned by the algorithm such that π is not a valid derivation from

E_{init} to E_{target}. This implies that at least one of the following holds:

1. $E_1 \neq E_{\text{init}}$, i.e., the path does not start at the initial expression.
2. $E_k \neq E_{\text{target}}$, i.e., the path does not end at the target expression.
3. There exists some $i \in \{1, \ldots, k-1\}$ such that E_{i+1} is not derivable from E_i via any valid transformation rule admissible under G.

We now show that none of these cases can occur under the design of DoVerifier:

- By construction, the algorithm initializes the search frontier with $\{E_{\text{init}}\}$, so the first element of any returned path is necessarily E_{init}.
- The algorithm terminates only upon finding an expression that is syntactically equal to E_{target}, so $E_k = E_{\text{target}}$.
- The algorithm only expands nodes via valid applications of transformation rules from the set $\mathcal{R}$, which includes do-calculus and standard probability rules. Each edge in the path corresponds to a rule in $\mathcal{R}$, and such rules are only applied if their preconditions (e.g., d-separation) hold in G.

Thus, any returned path must be a valid sequence of derivations from E_{init} to E_{target}, contradicting our assumption. Therefore, DoVerifier is sound. ■

Now we show DoVerifier is complete:

Proof. Suppose $E_{\text{init}} \vdash_G E_{\text{target}}$. Then by definition of $\vdash_G$, there exists a finite sequence of rule applications (i.e., a path in $S(E_{\text{init}})$) from E_{init} to E_{target}. Let the length of this shortest such sequence be d^*. Since $S(E_{\text{init}})$ is a well-defined, finite-branching graph (Theorem 3.1), BFS explores all nodes reachable from E_{init} up to depth d in increasing order of path length.

Therefore:

- If $d \geq d^*$, then E_{target} will be reached and returned as part of a valid proof sequence.
- If $d < d^*$, then E_{target} is not reachable within the bounded depth, and the algorithm correctly returns None.

Thus, the algorithm is complete up to the given depth d. ■

We can further argue that in the case of unbounded depth, if the algorithm terminates upon reaching an expression that is the same as its ancestors, our algorithm is still complete.

Proof. Assume the algorithm is modified so that whenever an expression E is expanded, any successor E' that is already present in the current path π from E_{init} to E is discarded. This prevents cycles. Since $\mathcal{L}_{\text{causal}}$ is **finite** for a fixed set of variables and the rule set $\mathcal{R}$ is finite, the search space $S(E_{\text{init}})$ is also finite. BFS with cycle avoidance will explore all reachable expressions in a finite number of steps. If E_{target} is reachable, BFS will eventually visit it (since BFS is exhaustive in a finite acyclic search space), and return a valid derivation. ■

E Practical Considerations

Fact E.1 (Complexity)**.** *The time complexity of BFS is $\mathcal{O}(b^d)$ where b is the maximum branching factor and d is the depth limit.*

While theoretically sound, practical implementations must consider several optimizations:

1. **Expression normalization** to avoid revisiting equivalent states (e.g., removing redundant conditions, standardizing variable order)
2. **Efficient d-separation testing** for determining rule applicability
3. **Memoization** of independence tests to avoid redundant graph operations
4. **Strategic ordering of rule applications** to potentially find solutions faster
5. **Bidirectional search** from both E_{init} and E_{target} to reduce the effective search depth

These optimizations preserve the theoretical guarantees while making the approach computationally feasible for practical use in evaluating causal reasoning in language models.

F Data Samples of Synthetic Data

To support the evaluation of causal inference methods, we construct synthetic datasets using directed acyclic graphs (DAGs) that encode assumed causal relationships among variables. Each DAG consists of nodes representing variables and directed edges representing direct causal influences. These graphs

serve as the basis for simulating both observational and interventional data.

The data samples are designed to validate derivations using do-calculus. Each example contains:

- A **DAG** representing the underlying relationships.
- A pair of probability expressions (E_a, E_b) where E_a is an interventional expression involving do-operators and E_b is an equivalent or simplified observational expression.
- A proof showing the sequence of do-calculus rules (Rule 1, Rule 2, Rule 3) applied to reduce E_a to E_b. These synthetic samples are not drawn from real-world distributions, but they adhere strictly to the independence constraints implied by the DAGs, ensuring the theoretical correctness of all derivations.

G Prompt Examples

To evaluate and guide language model performance on causal reasoning tasks, we designed a two-shot prompt that consists of: A set of instructions, two fully worked examples, a new query prompt for the model to solve in the same format.

```
## Instructions:
1. For each problem, identify the correct
    ↪ expression that represents the query
2. Draw the graphical representation as a
    ↪ text description of edges
3. Show your mathematical reasoning step by
    ↪ step
4. Provide a final yes/no answer
5. Keep your response concise and focused on
    ↪  the solution

## Examples:

Example 1:
Prompt: Imagine a self-contained,
    ↪ hypothetical world with only the
    ↪ following conditions, and without any
    ↪  unmentioned factors or causal
    ↪ relationships: Poverty has a direct
    ↪ effect on liking spicy food and
    ↪ cholera. Water company has a direct
    ↪ effect on liking spicy food. Liking
    ↪ spicy food has a direct effect on
    ↪ cholera. Poverty is unobserved. The
    ↪ overall probability of liking spicy
    ↪ food is 81%. The probability of not
    ↪ liking spicy food and cholera
    ↪ contraction is 13%. The probability
    ↪ of liking spicy food and cholera
    ↪ contraction is 17%. Is the chance of
    ↪ cholera contraction larger when
    ↪ observing liking spicy food?
Let V2 = water company; V1 = poverty; X =
    ↪ liking spicy food; Y = cholera

Expression: P(Y | X)
Graphical Representation: V1->X,V2->X,V1->Y,
    ↪ X->Y
Reasoning: P(X = 1, Y = 1)/P(X = 1) - P(X =
    ↪ 0, Y = 1)/P(X = 0)
P(X=1) = 0.81
P(Y=1, X=0) = 0.13
P(Y=1, X=1) = 0.17
0.17/0.81 - 0.13/0.19 = -0.44
-0.44 < 0
Final Answer: No

Example 2:
Prompt: Imagine a self-contained,
    ↪ hypothetical world with only the
    ↪ following conditions, and without any
    ↪  unmentioned factors or causal
    ↪ relationships: Poverty has a direct
    ↪ effect on liking spicy food and
    ↪ cholera. Water company has a direct
    ↪ effect on liking spicy food. Liking
    ↪ spicy food has a direct effect on
    ↪ cholera. Poverty is unobserved. For
    ↪ people served by a local water
    ↪ company, the probability of cholera
    ↪ contraction is 64%. For people served
    ↪  by a global water company, the
    ↪ probability of cholera contraction is
    ↪  66%. For people served by a local
    ↪ water company, the probability of
    ↪ liking spicy food is 50%. For people
    ↪ served by a global water company, the
    ↪  probability of liking spicy food is
    ↪ 45%. Will liking spicy food decrease
    ↪ the chance of cholera contraction?
Let V2 = water company; V1 = poverty; X =
    ↪ liking spicy food; Y = cholera.

Expression: E[Y | do(X = 1)] - E[Y | do(X =
    ↪ 0)]
Graphical Representation: V1->X,V2->X,V1->Y,
    ↪ X->Y
Reasoning: E[Y | do(X = 1)] - E[Y | do(X =
    ↪ 0)]
[P(Y=1|V2=1)-P(Y=1|V2=0)]/[P(X=1|V2=1)-P(X
    ↪ =1|V2=0)]
P(Y=1 | V2=0) = 0.64
P(Y=1 | V2=1) = 0.66
P(X=1 | V2=0) = 0.50
P(X=1 | V2=1) = 0.45
(0.66 - 0.64) / (0.45 - 0.50) = -0.39
-0.39 < 0
Final Answer: Yes

## Your Task:
Solve the following problem using the format
    ↪  above. Begin your response with "
    ↪ Solution:" and provide only the
    ↪ expression, graphical representation,
    ↪  reasoning, and final answer.
Prompt: {description}
```

H Formal Description of Feedback Loop

Given a causal graph $G = (V, E)$ (which may be LLM generated), an LLM generated expression $E_{\text{LLM}} = P(Y \mid$

$\mathrm{do}(X_1), \ldots, \mathrm{do}(X_k), Z_1, \ldots, Z_m)$, and **no access to the ground truth** E_{target}. Our goal is to compute a revised expression E'_{LLM} that is causally more valid (i.e., more likely to match E_{target}) using structural reasoning over G.

We do so by partitioning the conditioning set of E_{LLM} into intervention variables $\mathbf{X}_{\text{do}}$ and $\mathbf{Z}_{\text{obs}}$:

$$\mathbf{X}_{\text{do}} = \{X_1, \ldots, X_k\} \quad \mathbf{Z}_{\text{obs}} = \{Z_1, \ldots, Z_m\}$$

Then, for each variable $Z \in \mathbf{Z}_{\text{obs}}$, we test:

- **Mediator Detection:** If Z lies on a directed path from some ancestor $A \in \mathbf{Z}_{\text{obs}} \cup \mathbf{X}_{\text{do}}$ to outcome Y:

$$A \to \cdots \to Z \to \cdots \to Y$$

  Then, Z is a mediator, so we write a prompt to avoid conditioning on Z, as doing so may block part of the causal pathway and lead to underestimation of the effect.

- **Treatment Confounding:** If $Z \in \mathbf{Z}_{\text{obs}}$ is a *common cause* of both a treatment variable $X \in \mathbf{X}_{\text{do}}$ and the outcome Y, i.e., $Z \to X$ and $Z \to Y$, then Z is a *confounder*. In such cases, we suggest replacing Z with $\mathrm{do}(Z)$ when feasible, as intervening on Z may help eliminate confounding bias—particularly when front-door adjustment is applicable.

- **d-Separation Violation:** Let $\mathbf{W} = \mathbf{Z}_{\text{obs}} \backslash \{Z\} \cup \mathbf{X}_{\text{do}}$; if $X \not\perp Y \mid \mathbf{W}$, then we suggest conditioning on Z may bias the expression as it is not independent of Y given other variables $\mathbf{W}$.

I Alternative Metrics

Evaluation of causal expression generation has often relied on surface-level metrics such as exact string match, BLEU score, BERTscore, and token-level F1.

BLEU and Token-level F1 Fails for Causal Evaluation BLEU computes precision over n-grams between a candidate and reference string. In causal reasoning, it suffers from

Small expression length bias: Causal expressions are often short; hence, BLEU becomes unstable when evaluating < 10 token strings since higher-order n-grams vanish.

Syntactic Fragility: Expressions that are semantically equivalent but have different variable order get penalized.

Non-semantic penalties: BLEU may still reward inclusion of irrelevant variables if they overlap with the gold string, even if the overall expression is wrong.

Token-level F1 computes overlap between tokens, treating the expression as a bag of symbols. It however, still leads to multiple failure cases:

Ignores structure role of variables: F1 cannot distinguish $P(Y)$ from $P(Y \mid X)$ or $P(Y \mid \mathrm{do}(X))$. They call share some subset of overlapping tokens and will inflate the accuracy.

No notion of well-formedness: Syntactically expressions such as $P(X\ Y)$ or $Y \mid P(X)$ might have high F1 if they reuse common symbols despite being invalid.

No semantics: Conditioning vs intervention is completely ignored, a model can be rewarded for guessing the right letters, not the right logic.

Table 3 shows the average BLEU and token-level F1 score for each model evaluated on causal language tasks. We see that both BLEU and F1 lack a formal grounding in the semantics of causal inference. There is no transformation set $\mathcal{T}$ under which they define an equivalence class. In contrast, our symbolic verifier defines:

$$\phi_1 \equiv_G \phi_2 \iff \phi_1 \vdash_G \phi_2 \wedge \phi_2 \vdash_G \phi_1 \quad (15)$$

Thus, BLEU and F1 may disagree with formal correctness, and worse, may systematically overestimate the validity of incorrect outputs.

BERTScore Failure Cases BERTScore (Zhang et al., 2020) is a widely used metric that computes semantic similarity by aligning contextualized token embeddings from a pretrained BERT model. It is often promoted as a semantically aware alternative to BLEU. However, in the context of causal reasoning, BERTScore exhibits a distinct failure mode: it confuses lexical proximity for logical validity. Table 4 shows common failure cases where BERTscore assigns a high similarity score, even

Model	BLEU	Token-level F1
Llama-3.1-8B-Instruct (Grattafiori et al., 2024)	0.46	0.70
Mistral-7B-v0.1 (Jiang et al., 2023)	0.33	0.58
Llama-3.1-8B (Grattafiori et al., 2024)	0.36	0.57
Gemma-7b-it (Team et al., 2024)	0.19	0.55

Table 3: Average BLEU and token-level F1 scores for each model evaluated on CLadder.

LLM Output	Formal Label	Correct?	BERTScore F1
`P(Y | V1)`	`P(Y | X)`	No	0.91
`P(Y)`	`P(Y | X)`	No	0.91

Table 4: Incorrect model outputs with high BERTScore. While these expressions differ from the gold standard, BERTScore assigns high similarity, demonstrating its over-generosity in causal evaluation.

when they are not supposed to be equivalent expressions. Let $\phi_{\text{pred}}, \phi_{\text{gold}} \in \mathcal{L}_{\text{causal}}$ be causal expressions encoded as strings. BERTScore computes:

$$\text{BERTScore}(\phi_{\text{pred}}, \phi_{\text{gold}}) = \text{F1}_{\text{BERT}}(h_{\phi_{\text{pred}}}, h_{\phi_{\text{gold}}}) \tag{16}$$

where h_ϕ are contextual embeddings from a pre-trained BERT model. However, the model has no knowledge of causal semantics, independence structures, or the syntax of do-calculus. Tokens like `P`, `(`, `)` are close in embedding space regardless of their role in the logical formula. This results in BERTScore assigning high similarity to expressions that are semantically disjoint under the causal graph. Unlike DoVerifier, BERTScore lacks a soundness guarantee

$$\text{BERTScore}(\phi_{\text{pred}}, \phi_{\text{gold}}) > 0.9 \not\Rightarrow \phi_{\text{pred}} \equiv_G \phi_{\text{gold}} \tag{17}$$

This could become dangerous in high-stakes contexts, where plausible-looking causal statements may lead to incorrect conclusions when evaluated with BERTScore.

J Frequently asked questions

What problem does DoVerifier actually solve? DoVerifier addresses the gap between surface-form evaluation of causal reasoning in LLM outputs (e.g., string match, BLEU, BERTScore) and semantic correctness under causal inference rules. It checks whether a model's predicted causal expression is formally derivable from a given causal graph using do-calculus and probability rules, recovering correct answers that naive metrics miss.

Does DoVerifier require the ground truth answer? For evaluation, yes - the framework needs the correct expression to compare against. However, for feedback and self-correction, it can operate without the ground truth by checking the model's answer against the DAG and suggesting corrections.

Can't we just use the ID algorithm by Shpitser and Pearl (2008) to see if both are identifiable, and then compare? There are cases when the expressions are unidentifiable but can be simplified such as

$$E_1 = P(Y \mid \text{do}(X), \text{do}(W), Z)$$
$$E_2 = P(Y \mid \text{do}(X), Z)$$

under specific DAGs, which can be easily constructed to satisfy do-calculus rule 3.

How is DoVerifier different from Lean or other proof assistants? Lean is a general-purpose formal proof assistant used to verify mathematical theorems in a wide range of domains. It requires users to construct complete proofs in a formal language. DoVerifier is a domain-specific verifier for causal inference. It operates only on causal expressions, uses a fixed set of rules from do-calculus and probability theory, and performs automated proof search to determine equivalence between expressions given a causal graph. Users do not supply the proof steps; the system infers them automatically.