Privacy-preserving and Verifiable Causal Prescriptive Analytics

ZHAOYU WANG, Hong Kong University of Science and Technology, Hong Kong SAR
PINGCHUAN MA*, Hong Kong University of Science and Technology, Hong Kong SAR
ZHANTONG XUE, Hong Kong University of Science and Technology, Hong Kong SAR
YANBO DAI, Hong Kong University of Science and Technology, Hong Kong SAR
ZHENLAN JI, Hong Kong University of Science and Technology, Hong Kong SAR
SHUAI WANG, Hong Kong University of Science and Technology, Hong Kong SAR

Prescriptive analytics seeks to identify optimal interventions for achieving desired outcomes, with causal inference playing a pivotal role in assessing intervention impacts on complex systems. However, existing approaches frequently neglect critical data privacy considerations and provide no means to verify the integrity of their recommendations. These limitations hinder its adoption in high-stakes domains such as healthcare and finance.

In this paper, we introduce, zkCLEAR, a zero-knowledge proof (ZKP)-based **C**ausal Inference (**LEA**rning and **R**easoning) framework for privacy-preserving and verifiable prescriptive analytics. Our solution allows data owners or service providers to cryptographically prove the validity of prescriptive conclusions derived from causal analysis without disclosing sensitive source data or proprietary causal models. We develop a suite of ZKP-friendly causal operators to build efficient causal modules, including structure learning, parameter learning, probabilistic inference, and counterfactual reasoning. To optimize performance, we also introduce a workflow decomposition strategy to facilitate efficient proof generation for complex workloads. We demonstrate the utility of zkCLEAR through three real-world applications. The framework faithfully follows the behavior of non-ZKP counterparts, with moderate overheads for privacy and verifiability. Additionally, we evaluate its efficiency and scalability using real-world datasets. It shows up to a 35.1× speedup in proof generation time and a 214.5× reduction in proof size compared to current general-purpose ZKP systems.

CCS Concepts: • **Security and privacy** → **Domain-specific security and privacy architectures**; • **Mathematics of computing** → *Bayesian networks*; *Causal networks*; • **Computing methodologies** → *Causal reasoning and diagnostics.*

Additional Key Words and Phrases: Zero-knowledge proof, Causal inference, Prescriptive analytics

ACM Reference Format:
Zhaoyu Wang, Pingchuan Ma, Zhantong Xue, Yanbo Dai, Zhenlan Ji, and Shuai Wang. 2025. Privacy-preserving and Verifiable Causal Prescriptive Analytics. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 350 (December 2025), 30 pages. https://doi.org/10.1145/3769815

*Corresponding author.

Authors' Contact Information: Zhaoyu Wang, Hong Kong University of Science and Technology, Kowloon, Hong Kong SAR, zwangjz@cse.ust.hk; Pingchuan Ma, Hong Kong University of Science and Technology, Kowloon, Hong Kong SAR, pmaab@cse.ust.hk; Zhantong Xue, Hong Kong University of Science and Technology, Kowloon, Hong Kong SAR, zxueai@cse.ust.hk; Yanbo Dai, Hong Kong University of Science and Technology, Kowloon, Hong Kong SAR, ydai851@cse.ust.hk; Zhenlan Ji, Hong Kong University of Science and Technology, Kowloon, Hong Kong SAR, zjiae@cse.ust.hk; Shuai Wang, Hong Kong University of Science and Technology, Kowloon, Hong Kong SAR, shuaiw@cse.ust.hk.

ACM 2836-6573/2025/12-ART350
https://doi.org/10.1145/3769815

1 Introduction

Prescriptive analytics aims to empower decision-makers with actionable insights derived from data [22]. A promising approach to achieving this goal involves leveraging causal inference, which moves beyond mere correlation to effectively answer critical questions such as "what-if" (predicting outcomes under hypothetical scenarios) and "how-to" (determining necessary actions to achieve specific goals). Recently, the database community has increasingly adopted causal inference techniques to enhance transparency, interpretability, and accountability in data-driven decision-making [18, 19, 39, 49].

However, for causal inference to become widely trusted—especially in sensitive domains—two key challenges must be addressed: ensuring the *privacy* of underlying data and the *verifiability* of computed results. Although the database community has extensively explored privacy-preserving solutions for machine learning [23, 34, 54], query processing [21, 35, 47], and secure data sharing [58], these approaches have yet to be adapted to causal inference. To illustrate this critical gap, consider the following scenario:

Example 1.1. *A healthcare authority observes significant regional differences in patient recovery rates following a medical treatment. Analysts and stakeholders naturally seek explanations: Why do patients in Region A recover faster than those in Region B? By using a causal analysis tool such as Hyper [18], a plausible explanation might attribute this disparity to superior healthcare infrastructure, yet a biased provider could instead claim it results from the use of an expensive new drug they are promoting. Without proper verification, users must blindly trust explanations that could be intentionally misleading or inaccurate.*

Example 1.1 illustrates a important *socio-technical challenge* regarding the *verifiability gap* imposed by opaque data. In Example 1.1, privacy regulations prevent the disclosure of sensitive patient data, which in turn prevents external stakeholders from verifying the claim made by the healthcare provider. Addressing this gap requires technical solutions that enable verifiability without compromising privacy, which is currently lacking in toolchains.

In response, we introduce a novel framework zkCLEAR for *Privacy-preserving and Verifiable Causal Prescriptive Analytics*. zkCLEAR enables the healthcare provider to perform certified causal analysis and then generate an attestation to its causal conclusion. The attestation, in the form of a zero-knowledge proof (ZKP), validates that the provided explanation is computed accurately based on the cryptographically committed causal model and original data, without exposing confidential patient records. Users can independently verify this proof and thereby build justified confidence in the resulting decisions. To understand the role of zkCLEAR in real-life scenarios, we continue from Example 1.1.

Example 1.2. *Continuing from Example 1.1, the healthcare provider uses zkCLEAR to generate a zero-knowledge proof attesting that its causal explanation for regional disparities in patient recovery is computed faithfully from committed data and models. The healthcare authority, as an external verifier, verifies this proof locally. If the explanation is inconsistent with the committed data, the proof fails to verify and the verifier is alerted to potential misreporting.*

Technical Challenges. Despite advances in ZKP systems for privacy and integrity, adapting such cryptographic guarantees to causal inference scenarios like Example 1.1 presents several core challenges. First, many causal operations, such as ancestor identification, acyclicity checks, and d-separation tests, require expressive **graph-theoretic computation** that is difficult to encode efficiently in standard ZKP frameworks. These algorithms often involve data-dependent recursion and control flow, which can balloon circuit sizes when naively unrolled (§ 4.1). Second, modern

causal inference fundamentally relies on **real-valued computation**, yet most ZKP systems natively support only finite-field operations (§ 4.2). This gap becomes even more pronounced in machine learning tasks that underpin many causal models: both **ML training and inference** require intensive numerical computation (e.g., matrix operations) that are costly or impractical in conventional ZKPs (§ 4.3). Third, prescriptive analytics pipelines typically span multiple, interdependent steps, demanding not only end-to-end verifiability but also **modular proof generation** (§ 5.3). Attempting to generate a single monolithic proof for a complex workflow is often infeasible both in terms of scalability and flexibility.

Table 1. List of desired features in causal inference and their support in existing ZKP frameworks. ✗: not supported; ○: partial/impractical; ✓: supported.

	Constr. Sys.	ZKVM	ZKML	zkCLEAR
Real-valued Arithmetic	✗	○	✓	✓
Graph-theoretic Op.	✗	✗	✗	✓
ML Training	✗	✗	○	✓
ML Inference	✗	✗	✓	✓
Modular Proof Gen.	✗	✓	✗	✓

Existing Solutions. Table 1 summarizes the extent to which leading ZKP solutions, including constraint systems [6, 11], ZK virtual machines [30, 57], and ZKML frameworks [8, 15, 25, 52], address these essential requirements. Traditional constraint systems perform admirably at generic arithmetic but do not natively support real-valued computation, graph-theoretic primitives, or machine learning. ZKVMs offer gains in programmability and expressiveness, but remain inefficient for computation-intensive tasks. ZKML frameworks are specialized for neural network inference, while they cannot represent general graph operations or full-pipeline workflow decomposition. As noted in Table 1, none of these approaches comprehensively meet the combined demands of zero-knowledge prescriptive analytics (see § 7 for detailed discussion).

Our Solution. To overcome these challenges, we introduce zkCLEAR, the first specialized framework integrating ZKP with causal inference, specifically tailored for privacy-preserving and verifiable causal prescriptive analytics. zkCLEAR is built on two core technical contributions: First, we develop a novel suite of ZKP primitives optimized for core causal inference operations. We efficiently encode complex graph-theoretic computations—including ancestor identification, acyclicity checks, d-separation tests, and interventional graph transformations—by leveraging algebraic characterizations and randomized verification techniques. We also employ a fixed-point arithmetic scheme to robustly handle real-valued computations and integrate ZKP-friendly ML training and inference procedures. Second, we systematically integrate these operators into a modular, extensible framework that supports complex causal reasoning workflows. We construct a comprehensive set of causal analysis modules, including structure learning, parameter learning, probabilistic inference, and counterfactual reasoning, that compose the low-level operators into high-level causal reasoning capabilities. To enhance scalability, we design an innovative workflow decomposition strategy that breaks down complex causal inference workloads for efficient proof generation. We implement three representative prescriptive analytics applications and, through extensive empirical evaluation, demonstrate that zkCLEAR can faithfully and efficiently generate zero-knowledge proofs for a variety of real-world tasks. We also evaluate zkCLEAR under different configurations, including varying number of inputs and attributes to demonstrate its flexibility and adaptability to different use cases. Against leading ZKVM frameworks, zkCLEAR achieves up to 35.1× faster

proof generation and yields proofs up to 214.5× smaller across both operator- and module-level computation tasks. We summarize our contributions as follows:

- We introduce zκCLEAR, the first framework to integrate zero-knowledge proofs with causal inference for privacy-preserving and verifiable prescriptive analytics.
- We design a novel suite of ZKP primitives for causal operations and integrate them into a modular framework. Our decomposition strategies enable scalable and composable proof generation for complex analytics pipelines.
- We implement three representative prescriptive analytics applications under zκCLEAR and demonstrate their flexibility, correctness and efficiency. We conduct extensive experiments to evaluate the performance of zκCLEAR on various configurations and workloads.

The rest of the paper is organized as follows. § 2 reviews ZKPs and causal prescriptive analytics. § 3 outlines the system architecture, threat model, and application scope. § 4 introduces our ZKP operators. § 5 describes their composition into modules and workflows. We present evaluation in § 6, discuss related work in § 7, limitations in § 8, and conclude in § 9.
Availability. zκCLEAR is publicly available at [2].

2 Background

This section reviews the technical background: zero-knowledge proofs for privacy-preserving, verifiable computation, and prescriptive causal analytics, which zκCLEAR is designed to support.

2.1 Zero-Knowledge Proofs (ZKPs)

Let $\mathcal{R}$ be an efficiently computable binary relation consisting of pairs (s, w), where s is a statement and w is a witness. Let $\mathcal{L}$ be the language associated with $\mathcal{R}$, i.e., $\mathcal{L} = \{s \mid \exists w \text{ s.t. } R(s, w) = 1\}$. A ZKP for $\mathcal{L}$ is a cryptographic protocol that allows a prover P to convince a verifier V that a public statement $s \in \mathcal{L}$ (that is, s is satisfiable with some valid witness) without revealing the private witness w. A ZKP captures not only the truth of a statement $s \in \mathcal{L}$, but also that the prover possesses a witness w to this fact.

ZKP protocols are generally categorized into two types: interactive ZKPs [20] and non-interactive ZKPs [7]. In this paper, we focus on non-interactive ZKPs, which consolidate the entire proof into a single message. This single-message approach eliminates the need for simultaneous online interaction, allowing the verifier to check the proof at any time, even if the prover is offline. This feature makes them suitable for scenarios where both parties cannot be online at the same time (e.g., in blockchain applications). Formally, a non-interactive ZKP for an NP relation $\mathcal{R}$ consists of a triple of polynomial time algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$:

(1) $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: Setup public parameters pp for a security parameter λ.
(2) $\mathsf{Prove}(\mathsf{pp}, s, w) \to \pi$: If $\mathcal{R}(s, w) = 1$, output a proof π.
(3) $\mathsf{Verify}(\mathsf{pp}, s, \pi) \to \{0, 1\}$: Verify a proof π for the statement s.

The algorithms above should satisfy the following properties.

(1) *Completeness*: An honest prover can always convince an honest verifier when the statement is true;
(2) *Soundness*: A dishonest prover cannot convince an honest verifier of a false statement except with negligible probability;
(3) *Zero-Knowledge*: The verifier learns nothing about the secret witness beyond the validity of the statement.

Next, we further narrow down our focus to non-interactive ZKPs that are based on *succinct* arguments of knowledge, a.k.a. ZK-SNARKs [7]. ZK-SNARKs offer succinct proof size and sublinear

verification times relative to the complexity of the relation R, which makes them practical for real-world applications. In ZK-SNARKs, the computation is typically represented as a *circuit* C, which consists of two parts: computation function f and constraint relations $\mathcal{R}$. Given some public input x and secret input w, the circuit C computes the output $y = f(x, w)$. The constraint relations $\mathcal{R}$ are satisfied if and only if the output y is correctly computed by $f(x, w)$. The formal definition of a circuit is as follows.

Definition 2.1 (Circuit). *A* circuit C *is a formal representation of a computational process defined as the tuple*

$$C = (f, \mathcal{R}),$$

where:

- $f : \mathcal{X} \times \mathcal{W} \to \mathcal{Y}$ *is the computation function that maps a public input* $x \in \mathcal{X}$ *and a secret input* $w \in \mathcal{W}$ *to an output* $y \in \mathcal{Y}$*;*
- $\mathcal{R}$ *is a set of constraint relations, typically expressed as a system of polynomial equations defined over a large prime field.*

Typically, $\mathcal{R}$ is defined over a large prime field supporting only addition and multiplication. To ease the development of ZKP circuits, many ZKP frameworks have been developed, which abstracts these low-level cryptographic constructs [6, 11].

2.2 Prescriptive Analytics and Causal Inference

Data analysis techniques are commonly categorized into three types [16]: descriptive, predictive, and prescriptive analytics—each addressing a distinct decision-making questions. *Descriptive analytics* answers the "what happened" question by summarizing historical data to reveal past patterns. *Predictive analytics* forecasts "what will happen?" using historical trends and probabilistic models. *Prescriptive analytics* makes one step further to answer "what-if" and "how-to" questions. Typically, this capability is achieved by a combination of techniques from *causal inference*.

Causal inference offers a structured approach to understanding cause-effect relationships within complex systems [13]. As shown in Fig. 1, it typically unfolds in four fundamental steps: structure learning, parameter learning, probabilistic inference, and counterfactual inference. Given a dataset as input, structure learning identifies the causal graph that represents the qualitative cause-effect relationships among the variables. With the causal graph and the dataset, parameter learning estimates the parameters of the causal model which quantitatively describes the relationships between the variables. Then, with the quantitative causal model, probabilistic inference computes the conditional probabilities of the variables in the system, similar to traditional machine learning tasks, though with a more principled causality-based approach. Finally, with these as a foundation, counterfactual inference estimates the potential outcomes under hypothetical interventions or alternative past events, which direct answers the "what-if" and "how-to" questions. In practice, users may manually specify the causal graph or quantitative causal model based on their domain knowledge. In that sense, some steps may be skipped.

3 Approach

This section presents the design of zkCLEAR, which uses a layered architecture to build complex causal workflows from low-level, ZKP-friendly primitives. It enables efficient encoding of core causal operations via specialized ZKP operators and supports scalable composition into end-to-end analytics.

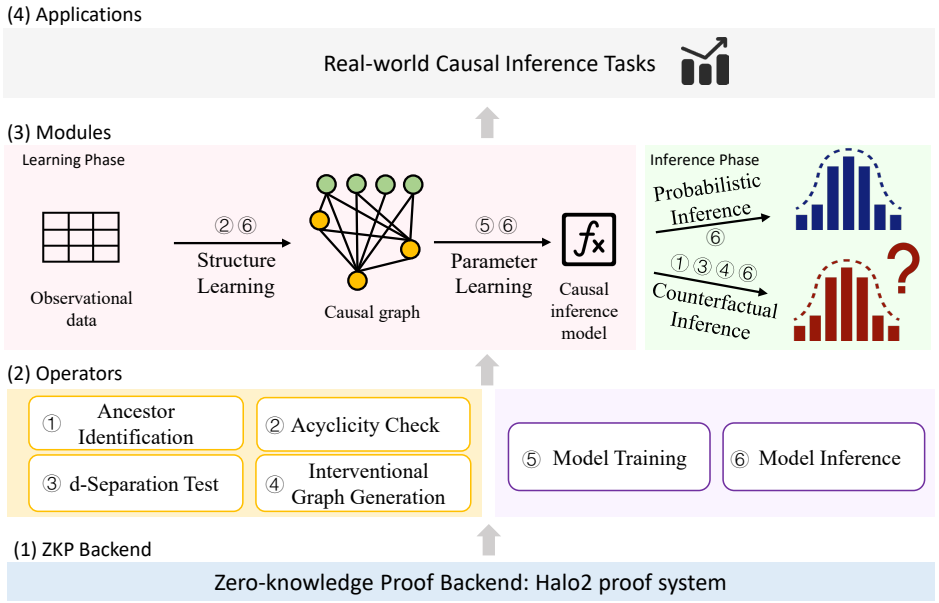


Fig. 1. Framework overview.

3.1 Technical Pipeline

Fig. 1 illustrates the technical pipeline of zkCLEAR. Overall, our framework features a bottom-up design that integrates ZKP with causal inference. At the lowest level, we design a suite of elementary operators using ZKP systems, which are the building blocks of our framework. These operators form important primitives for handling graph-theoretic operations, arithmetic computations, and machine learning tasks. We then build a set of causal modules on top of these primitives to enable the key tasks we have identified in § 2.2. Finally, we build advanced causal applications through a combination of these modules to support real-world prescriptive analytics tasks. Given a specific causal analysis task, we further provide a workflow decomposition strategy to break down the task into smaller sub-tasks, which can be executed in parallel to improve the efficiency of proof generation. Below, we provide a brief overview of our ZKP backend as well as these components.

ZKP Backend. Our system is built within the halo2 ecosystem [11] (blue box in Fig. 1), which provides a general-purpose ZKP circuit construction framework based on PLONKish arithmetization [17]. Compared to R1CS-based systems (e.g., Circom [6]), the PLONKish approach supports custom gates and lookup arguments which enable the efficient and flexible implementation of complex arithmetic operations, including non-linear functions. Our toolchain integrates *halo2-lib* [3], a Rust library with a high-level interface for circuit construction, and ezkl [25], an optimized library tailored for deep neural network (DNN) inference within halo2. *halo2-lib* is primarily used to implement graph and numerical operators, while *ezkl* is leveraged to enable DNN inference.

Operators (§ 4). As the most fundamental basis of our framework, the causal operator layer supplies three families of primitives: graph operators (for ancestor identification, acyclicity checks, d-separation tests, and interventional graph construction; yellow box in Fig. 1), numerical operators (for basic arithmetic operations in prime fields; omitted in Fig. 1 since they are used internally by ML operators), and ML operators (for machine learning model training and inference; purple box in Fig. 1). Together, these operators form the low-level building blocks needed for causal inference.

Modules (§ 5.1). Building on the foundation of the causal primitives, the causal module layer provides a suite of core functionalities that span the fundamental phases of causal inference, including structure learning and parameter learning at the offline learning stage (red box in Fig. 1), as well as probabilistic and counterfactual inference at the online inference stage (green box in Fig. 1).

Applications (§ 5.2). The final layer of our framework, the causal applications layer (gray box in Fig. 1), demonstrates the practical utility of zkCLEAR in addressing real-world challenges. This layer showcases three concrete implementations of advanced causal inference workflows built upon the modules above.

To illustrate how end users interact with zkCLEAR in practice, we provide a visual walkthrough in Fig. 2, based on Example 1.2. The prover (e.g., a healthcare provider) prepares private inputs (e.g., patient health records, causal model) and commits to them using cryptographic hashing. The prover then executes the causal inference workflow using zkCLEAR, producing both a causal conclusion (e.g., "better healthcare infrastructure improves recovery") and a ZK proof attesting to its correctness. The verifier (e.g., a healthcare authority) locally verifies the result and proof against the public commitments and is convinced of the causal conclusion.

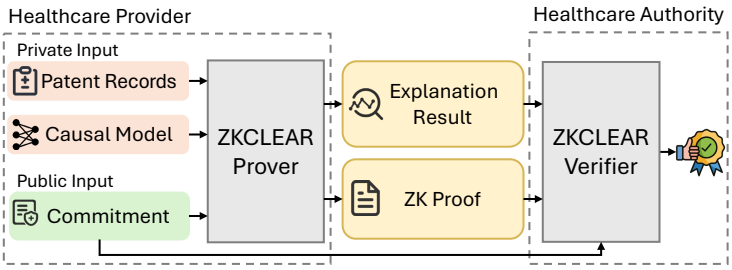


Fig. 2. Visual walkthrough of zkCLEAR.

3.2 Threat Model and Application Scope

zkCLEAR mitigates data privacy breaches in causal analysis by employing ZKP. Below, we analyze the threat model (i.e., potential privacy threats) and the application scope of zkCLEAR.

Threat Model. The assets of interest are versatile in our framework. On one hand, it can be the raw data used to learn the causal model, or the data passed to the causal model for causal reasoning. For example, in Example 1.1, the patients' health records remain confidential to meet regulations and ethical concerns. On the other hand, it can also be the causal model itself, which is often proprietary and confidential. In Example 1.1, the causal model represents the learned relationships between treatments and outcomes, which may require confidentiality to protect intellectual property. It is also possible that none of these assets are of interest, but the integrity of the computation process is the main concern. In this case, the prover may not even have any private data, but the prover still needs to ensure that the computation process (e.g., the computation of generating causal explanations) is correct, which is a common setting for blockchain applications.

In all these cases, the integrity of data/model is guaranteed by cryptographic commitments (e.g., hash functions) encoded in the ZKP circuit, and the integrity of computation is guaranteed by ZKP itself. The security of the ZKP and the cryptographic commitments depends on the underlying ZKP system. In zkCLEAR, we inherit the security guarantees (completeness, knowledge soundness, and zero-knowledge) of the halo2 proof system using KZG scheme [28], which relies on the assumption that the trusted setup is performed honestly. It is worth noting that the trusted setup is a one-time

universal setup, which means that once established, it can be reused for any circuits. Users can rely on a publicly coordinated and audited trusted setup, such as the one conducted by the Privacy Scaling Explorations project, to eliminate the need for application-specific setups or additional trust assumptions. Besides, in the construction of some operators, we rely on a trusted source of randomness to generate the random numbers used in the ZKP circuit or an additional round of interaction between the prover and the verifier.

Application Scope. zkCLEAR is designed as a general-purpose framework for ZKP-based prescriptive analytics, with a modular architecture that supports extensibility to a broad range of causal analysis tasks and algorithms. Specifically, zkCLEAR can be adapted to express statistical estimands used in leading probabilistic causal inference systems, such as CaRL [49], CauSumX [55], and Causal Prescription Rule [33], in addition to the three concrete applications demonstrated in § 5.2. Currently, zkCLEAR operates on single-table inputs. For relational data settings, we anticipate that integrating zkCLEAR with ZK-SQL [35] could enable scalable ZK causal analysis across relational schemas. That said, certain causality paradigms fall outside the current design scope of zkCLEAR. In particular, frameworks grounded in database causality [39, 41, 42] or intervention-based formal models [48] are not directly supported, as they differ from the probabilistic foundations that zkCLEAR builds upon.

4 Building Block

In this section, we present the key building blocks in zkCLEAR.

4.1 Graph Operator

A major distinction of causal inference from traditional machine learning is its dependency on causal graphs, which inherently involves complex manipulations, such as traversals, pathfinding, and structural transformations. These manipulations are trivial in traditional programming paradigms but become very challenging in ZKP systems.

Challenges. Directly implementing graph algorithms within ZKP systems poses significant technical challenges due to presence of complex control flow and data-dependent operations [27]. Specifically, conditional logic (e.g., branches) and dynamic traversal patterns (e.g., breadth-first search) cannot directly map to fixed arithmetic circuits, as execution paths must remain input-independent to preserve zero-knowledge guarantees. While standard techniques for handling conditional logic in ZKPs exist, such as boolean selectors that compute all branches concurrently and then select the appropriate outcome, they introduce significant computational overhead. This issue becomes even more pronounced in the case of graph traversal algorithms, as each potential traversal path must be encoded in the circuit regardless of whether it's actually taken during execution. This requirement leads to exponential growth in circuit size and verification complexity, making naive implementations impractical for real-world ZKP applications. To address these issues, we investigate the common workflow in causal inference and propose to optimize the following graph operations.

4.1.1 Ancestor Identification. Given a graph, ancestor identification aims to determine all nodes that are the ancestors of a specified node. In causal inference, this operation provides key information in identifying the backdoor paths and determining the necessary variables to control for cause-effect estimation.

Basic Idea. To effectively express the ancestor identification operation in ZKP, we first need an abstract, structure-agnostic representation of reachability in the graph. Graph theory provides such a representation through the adjacency matrix power series. Let A represent the adjacency matrix of the causal graph, where n the number of nodes. The reachability matrix, denoted as M, is then

calculated as:

$$M = \sum_{k=0}^{n-1} A^k.$$

This formulation can express the reachability between any two nodes in the graph as it includes paths of all lengths from 1 up to $n-1$, the maximum length of a simple path in a graph with n nodes. Consequently, when $M_{ij} > 0$, it signifies that node j is reachable from node i, indicating that i is an ancestor of j.

Building upon this concept, once the reachability matrix M is computed, identifying the ancestors of any node i becomes efficient: it is sufficient to examine the non-zero entries in the i-th column of M. This method is particularly advantageous when multiple ancestry queries need to be answered on the same causal graph, as it allows for fast lookups after a single computation of M.

Improved Solution. While the approach above provides a solid theoretical foundation, its practical implementation within ZKP systems imposes substantial overheads with $O(n^4)$ operations in the constraints. To address this challenge, we turn to linear algebra for a more elegant formulation.

A key insight emerges when we examine the properties of adjacency matrices in causal DAGs. For the adjacency matrix A of a causal DAG, we can prove that $A^k = 0$ for $k \geq n$. This property holds because A^k_{ij} represents the number of paths of length k from node i to j. In a directed acyclic graph with n nodes, any simple path can visit each node at most once, meaning the longest possible simple path contains at most $n-1$ edges. Any path of length $k \geq n$ would necessarily contain a cycle, which contradicts the acyclic property of a DAG. Therefore, we have $A^k = 0$ for $k \geq n$, and consequently, the reachability matrix can also be expressed as an infinite sum: $M = \sum_{k=0}^{n-1} A^k = \sum_{k=0}^{\infty} A^k$.

Drawing from the theory of matrix geometric series, we know that for any matrix A where $\lim_{k\to\infty} A^k = 0$, the infinite sum $\sum_{k=0}^{\infty} A^k$ converges to $(I-A)^{-1}$. Thus, we can express the reachability matrix in a more compact form:

$$M = \sum_{k=0}^{n-1} A^k = \sum_{k=0}^{\infty} A^k = (I-A)^{-1}$$

While this mathematical transformation significantly reduces theoretical complexity of representing the reachability matrix, it simultaneously introduces a practical challenge: how to efficiently perform matrix inversion within the constraints of ZKP systems. Rather than directly encoding the entire matrix inversion computation, which would likely be prohibitively expensive in terms of constraint generation, we adopt a verification-centric approach. In this strategy, the reachability matrix M is first computed outside the ZKP system. Then, within the ZKP framework, we focus on verifying the correctness of this computed matrix M as the inverse of $(I-A)$. This verification relies on the fundamental property of matrix inversion: for a matrix M to be the inverse of $(I-A)$, their product must equal the identity matrix, i.e., $M \cdot (I-A) = I$. The elegance of this approach lies in shifting the computational burden from complex matrix inversion within ZKP to the simpler task of verifying matrix multiplication, which significantly reduces the overall constraint complexity.

To further optimize the verification, we integrate Freivalds' algorithm, a probabilistic method designed for efficient verification of matrix multiplication. Given matrices A, B, and their claimed product C, Freivalds' algorithm operates by randomly selecting a vector r, and then checking if the equality $Cr = A \cdot Br$ holds. If the equality holds, then $A \cdot B = C$ with a high probability. This approach reduces the verification complexity of verifying matrix multiplication from $O(n^3)$ to $O(n^2)$; we use this to verify matrix multiplication $M \cdot (I-A) = I$. Furthermore, we also employ Freivalds' algorithm as a standard technique throughout our primitive implementations whenever matrix multiplication verification is required. Finally, we consolidate the circuit as $C_{\textit{Ancestor-Identification}}$ in Fig. 3.

Circuit $C_{\textit{Ancestor-Identification}}$

Private Witness: DAG adjacency matrix $A \in \{0,1\}^{n\times n}$.
Output: Reachability matrix M of the DAG.
Computation function $\mathcal{F}$: Compute reachability matrix $M = (I - A)^{-1}$.
Witness generation: Generate the random vector $\mathbf{r}$.
Constraint relation $\mathcal{R}$: Verify $I = M \cdot (I - A)$ using $\mathbf{r}$ and Freivalds' method.

Fig. 3. Circuit for ancestor identification.

4.1.2 Acyclicity Check. An acyclicity check determines whether a given graph is acyclic, i.e., whether it qualifies as a valid causal graph. This check is a prerequisite for many causal inference tasks. In non-ZKP systems, acyclicity is typically verified using depth-first or breadth-first search to detect cycles. However, as with ancestor identification, these methods involve substantial data-dependent control flow that is challenging to work efficiently in ZKP systems.

To circumvent these difficulties, our methodology ($C_{\textit{Acyclicity-Check}}$ in Fig. 4) adopts a two-phase process. First, we perform the acyclicity check outside the ZKP framework, denoting the result as $\Phi_{\text{Acyclicity}}$, which we currently treat as untrusted. In the second phase, we verify within the ZKP framework whether the input graph is indeed acyclic (or cyclic), based on the provided $\Phi_{\text{Acyclicity}}$ claim.

Validate Acyclic Claims. When $\Phi_{\text{Acyclicity}}$ is true, we aim to verify that the input graph is indeed acyclic. This is achieved by a useful mathematical property of acyclic graphs: any acyclic graph can be permuted into a strictly upper triangular matrix. Since the proposition is well-known, we omit the proof here. With this property, the key idea is to find such a permutation matrix P that transforms the adjacency matrix A into a strictly upper triangular matrix B via the equation $B = P^\top AP$. Again, this step can be performed outside the ZKP framework. The permutation matrix P is a square binary matrix where each row and each column has exactly one entry of 1 and all other entries are 0. Within the ZKP constraint, we only need to verify that the matrix multiplication $P^\top AP$ yields a strictly upper triangular matrix. In summary, the constraint includes three checks: (1) *Permutation Validity:* Verify that P is a valid permutation matrix by enforcing the constraints $\sum_{i=1}^{n} P_{ij} = 1$ and $\sum_{j=1}^{n} P_{ij} = 1$ for all i, j; (2) *Matrix Multiplication:* Confirm that $B = P^\top AP$ using Freivalds' algorithm; (3) *Upper Triangular Structure:* Enforce that $B_{ij} = 0$ for $i \geq j$. All these checks involve data-independent arithmetic operations and are efficient in ZKP systems.

Validate Cyclic Claims. When $\Phi_{\text{Acyclicity}}$ is false, we need to verify that the input graph is indeed cyclic. For this case, we can leverage the fact that a cyclic graph must contain at least one cycle path. A cycle path $(v_1, v_2, \ldots, v_k, v_1)$ by definition is a sequence of nodes ($k \leq$ #node) where the first and last nodes are the same. Therefore, similar to the acyclic case, we can identify a cycle path outside the ZKP framework and validate it in the input graph under ZKP. Since the length of the cycle path is bounded, we take a fixed length k for the edges (i.e., a pair of adjacent nodes) $(v_i, v_{i+1}), \ldots, (v_k, v_{k+1}), (\epsilon, \epsilon), \ldots$, where ϵ is a dummy node for padding. The constraint validates that (1) all edges in the cycle path exist in the graph, i.e., $A_{v_i v_{i+1}} = 1$; (2) the tail of one edge matches the head of the next, i.e., $v_{i+1} = v_{i+2}$; and (3) the tail and head of the cycle path are the same, i.e., $v_{k+1} = v_1$.

4.1.3 D-Separation Test. In addition to standard graph-theoretic properties, causal inference often requires checking d-separation between two sets of nodes. D-separation is a key graphical criterion that allows us to infer conditional independence relationships from a causal graph. Specifically,

Circuit $C_{Acyclicity\text{-}Check}$

Private Witness: DAG adjacency matrix $A \in \{0,1\}^{n\times n}$.
Public Input: Randomness seed r_v.
Output: Boolean indicating whether the graph is acyclic or not.
Computation function $\mathcal{F}$: Check whether the graph is acyclic.
Witness generation:

(1) Generate the random vector $\mathbf{r}$.
(2) If the graph is acyclic, compute the permutation matrix P and the transformed strictly upper triangular matrix $B = P^\top AP$.
(3) If the graph is cyclic, compute the cycle path $(v_1, v_2, \ldots, v_k, v_1)$.

Constraint relation $\mathcal{R}$:

(1) If the graph is acyclic:
  (a) Verify P is a permutation matrix by enforcing the constraints: $\sum_{i=1}^{n} P_{ij} = 1$ and $\sum_{j=1}^{n} P_{ij} = 1$.
  (b) Verify $B = P^\top AP$ using the random vector $\mathbf{r}$ and Freivalds' method.
  (c) Verify B is strictly upper triangular by enforcing the constraints: $B_{ij} = 0$ for $i \geq j$.
(2) If the graph is cyclic, check the validity of the cycle path by enforcing the constraints: $A_{v_i v_{i+1}} = 1$ for $i = 1, \ldots, k-1$ and $A_{v_k v_1} = 1$.

Fig. 4. Circuit for acyclicity checks.

given two sets of nodes X and Y and a conditioning set Z, we say that X and Y are d-separated by Z if all paths between them are "blocked" by nodes in Z. This condition implies that X and Y are conditionally independent given Z.

In traditional (non-ZKP) systems, d-separation is typically verified through recursive graph traversal algorithms, which enumerate all paths between X and Y and check for blockage by Z. However, these methods involve data-dependent control flow and recursion, which are costly in ZKP systems. To make d-separation checking efficient with ZKP constraints, we aim to avoid recursion and non-uniform branching, and instead, rely on algebraic operations that can be efficiently verified. However, a direct algebraic formulation of d-separation is nontrivial, since the notion of "path blockage" in causal graphs deviates from standard graph-theoretic definitions and introduces structural subtleties.

To address this, we adopt a well-established graph-theoretic reduction based on moral graphs and ancestral subgraphs [32]. The key insight is captured by the following theorem.

Theorem 4.1. *Variables X and Y are d-separated by Z if and only if no path exists in the undirected ancestral moral graph with Z removed.*

As detailed in $C_{d\text{-}Separation}$ (see Fig. 5), the ZKP implementation of this theorem involves four key steps:

(1) **Compute the ancestral graph.** We begin by identifying all ancestors of nodes in $X \cup Y \cup Z$. This is done by computing the reachability matrix M of the DAG G using our ancestor identification circuit. Using M, we extract the ancestral subgraph $G_{ancestor(X\cup Y\cup Z)}$ by masking out all non-ancestor nodes in G.
(2) **Construct the moral graph.** We convert the ancestral DAG into an undirected graph by adding edges between all pairs of nodes that are parents of a common child. This "moralization" process can be efficiently achieved by computing $G' = G_{ancestor} \cdot G_{ancestor}^\top$, and binarizing the result to obtain an adjacency matrix for the moral graph using Freivalds' method.

Circuit $C_{d\text{-}Separation}$

Private Witness: DAG adjacency matrix $G \in \{0,1\}^{n \times n}$.
Public Input: Node sets A, B, and S for d-separation tests.
Output: Boolean indicating whether the two sets of nodes A and B are d-separated by Z or not.
Computation function $\mathcal{F}$:

(1) Identify all ancestors of nodes in $X \cup Y \cup Z$, and compute the ancestral graph $G_{ancestor(X \cup Y \cup Z)}$ by masking out all non-ancestor nodes in G.
(2) Construct the moral graph G' of $G_{ancestor(X \cup Y \cup Z)}$ by computing $G' = G_{ancestor(X \cup Y \cup Z)} \cdot G^{\top}_{ancestor(X \cup Y \cup Z)}$ and binarizing it.
(3) Remove all nodes in the set Z from the moral graph G' by masking them from the moral graph's adjacency matrix
(4) Compute the reachability matrix of the moral graph using Floyd-Warshall algorithm and check whether there is a path between X and Y.

Constraint relation $\mathcal{R}$:

(1) Verify $G_{ancestor(X \cup Y \cup Z)} = G \circ M_{ancestor(X \cup Y \cup Z)}$ where $M_{ancestor(X \cup Y \cup Z)}$ is non-ancestor mask.
(2) Verify $G' = G_{ancestor(X \cup Y \cup Z)} \times G^{\top}_{ancestor(X \cup Y \cup Z)}$ using the random vector $\mathbf{r}$ and Freivalds' method.
(3) Verify $G'' = G' \circ M_S$ where M_S is S-node mask.
(4) Verify the Floyd-Warshall algorithm.

Fig. 5. Circuit for D-Separation tests.

(3) **Remove conditioning nodes.** Nodes in the conditioning set Z are removed from the moral graph by masking them from the moral graph's adjacency matrix G'.
(4) **Check for connectivity.** Finally, we test whether any paths exist between nodes in X and Y within the modified moral graph. Since the graph is now undirected and may contain cycles, we cannot apply the matrix inversion trick used earlier. Instead, we use the Floyd-Warshall algorithm (Algorithm 1) to compute the transitive closure of the moral graph in $O(n^3)$ time using only fixed and ZKP-friendly operations.

Algorithm 1: Floyd-Warshall Algorithm

Input: Adjacency matrix $G \in \{0,1\}^{n \times n}$ of the causal graph.
Output: Transitive closure matrix T, where $T[i][j] = 1$ if there exists a path from i to j.

```
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do  T[i][j] ← T[i][j] ∨ (T[i][k] ∧ T[k][j]) ;
    end
end
return T
```

4.1.4 Interventional Graph Generation. Besides checking graphical properties, causal inference often involves manipulating the graph structure to simulate hypothetical scenarios. For example, when estimating the causal effect of a treatment on an outcome using do-calculus, it is necessary to derive the interventional graph. Informally, given a causal graph G and a target node v to intervene on, the interventional graph G^{intv} is constructed by removing all incoming edges to v. This modification reflects that the value of v is now determined exogenously by the intervention,

Circuit $C_{Linear\text{-}Training}$

Private Witness: Training data matrix $X \in \mathbb{R}^{n \times d}$, label vector $y \in \mathbb{R}^n$.
Public Input: Regularization parameter λ.
Output: Parameter vector $\beta \in \mathbb{R}^d$.
Computation function $\mathcal{F}$: Compute the closed-form solution of linear regression model.

(1) Compute $A \leftarrow X^\top X + \lambda I_d$.
(2) Compute $\beta \leftarrow A^{-1} X^\top y$ and return β.

Constraint relation $\mathcal{R}$:

(1) Verify the correctness of the matrix multiplication $X^\top X$ using the random vector $\mathbf{r}$ and Freivalds' method.
(2) Verify $A = X^\top X + \lambda I_d$ and $X^\top y = A\beta$.

Fig. 6. Circuit for linear regression training.

rather than by its usual causes. To do so, we simply zero out the v-th column of the adjacency matrix A, i.e., $A_{:,v} \leftarrow 0$. As this operation is straightforward in ZKP systems, we omit the circuit ($C_{Interv.\text{-}Graph}$) details.

4.2 Arithmetic Operator

As mentioned in § 2.1, ZKP systems operate over finite fields with addition and multiplication as the basic arithmetic operations. However, real-world causal inference workloads often involve real-valued data and computations. To bridge this gap, we leverage a fixed-point representation of real numbers within finite fields and implement basic arithmetic operations that are compatible with this representation.

Following previous works [36, 53], we represent a real number $\hat{x} \in \mathbb{R}$ as a field element $x \in \mathbb{F}_p$ via a fixed-point representation. This representation is parameterized by a scale variable s, which determines the fractional bit length. Given a prime p and a scale s, we can represent a real number $\hat{x}$ as a field element x in $\mathbb{F}_p$ using the following mapping:

$$x = \mathsf{Quantization}(\hat{x}, p, s) = \lfloor \hat{x} \cdot 2^s \rfloor \bmod p \tag{1}$$

Then, we can recover the original real number $\hat{x}$ from the field element x using the following mapping:

$$\hat{x} = \mathsf{Dequantization}(x, p, s) = \frac{x - c \cdot p}{2^s} \tag{2}$$

where $c = \mathbf{1}_{\{x > (p-1)/2\}}$. The quantization scheme allows encoding signed real numbers $\hat{x} \in \left[-\frac{p-1}{2}, \frac{p-1}{2}\right]$ at the precision of 2^{-s}.

With the quantization scheme, a real number $\hat{x}$ can be represented by a couple of integers (x, s) within the finite field $\mathbb{F}_p$. We further simulate the basic arithmetic operations over the fixed-point representation of real numbers using a set of addition/multiplication operators on (x, s). The operators include addition, subtraction, multiplication, and division, with support for comparison operations. All numerical computations in zkCLEAR use this fixed-point representation and its arithmetic operators.

4.3 ML Operator

During causal inference, we often need to perform ML model training and inference to quantitatively estimate causal effects on top of qualitative causal relationships constructed by graph operators. While it is possible to implement ML operators using the basic arithmetic operators described in § 4.2, intensive matrix operations in modern ML models impose significant overheads [8]. In zkCLEAR, we aim to provide training and inference capabilities, respectively.

Training. ML training, in general, is performed by optimizing a loss function over a dataset using gradient descent or other optimization techniques. This process inherently involves iterative updates to model parameters based on the computed gradients [52]. However, causal inference commonly relies on linear modeling assumptions, a special case that enables closed-form solutions. This linearity assumption is widely adopted in causal inference, especially when identifiability are prioritized. Crucially, the availability of closed-form solutions makes training significantly more efficient and ZKP-friendly, which is important in ZKP systems where proof generation costs can be substantial. Given a training dataset $X \in \mathbb{R}^{n \times d}$ and a label vector $y \in \mathbb{R}^n$, the parameter vector $\beta \in \mathbb{R}^d$ of the linear model is computed as follows:

$$\beta = (X^\top X + \lambda I_d)^{-1} X^\top y \tag{3}$$

where λ is a regularization parameter. The closed-form solution allows us to compute the model parameters in a single pass over the data. Since $X^\top X + \lambda I_d$ is always invertible, this closed-form solution is universal for all datasets. Similar to the approach of matrix inverse verification in $C_{Ancestor\text{-}Identification}$, it can be efficiently verified using Freivalds' algorithm. The circuit $C_{Linear\text{-}Training}$ is detailed in Fig. 6.

Inference. In causal inference, ML inference is involved in two ways. First, we employ a pretrained DNN model to infer the causal graph structure. Second, we use the trained linear model to estimate the causal effect of the treatment on the outcome. The latter, as a linear model, can be trivially expressed as a multiplication of the input matrix X and the parameter vector β. The former could be more complex when large DNN model is used. In zkCLEAR, we utilize the ezkl framework [25], a specialized and well-optimized ZKML framework, to synthesize circuits for pretrained DNN model inference. The circuits for both linear regression inference $C_{Linear\text{-}Inference}$ and DNN model inference $C_{DNN\text{-}Inference}$ are omitted here for brevity.

4.4 Remarks

A key design principle across our ZKP operators is to offload as much computation as possible to the external environment, and limit the role of the ZKP circuit to merely verifying the correctness of externally computed results. This strategy fundamentally distinguishes zkCLEAR from general-purpose ZKP systems, which often attempt to encode the entire computation in circuit, leading to prohibitive overhead for real-world causal tasks. Instead of directly computing complex operations like matrix inversion or structure learning within the circuit, we shift these to an untrusted preprocessing phase. The ZKP circuit then verifies their correctness using lightweight algebraic checks, or that a claimed permutation leads to an upper-triangular form. These verification tasks are not only cheaper in terms of circuit complexity than their computational counterparts, but also far more compatible with ZKP system constraints. This principle of *verification-over-computation* pinpoints our entire operator layer, and addresses the challenges noted in § 1. By doing so, zkCLEAR achieves a favorable trade-off between expressiveness and efficiency.

5 Putting It All Together

With ZKP operators in § 4, we show how to put them together to form zkCLEAR as an efficient ZKP causal inference framework.

5.1 Module

As introduced in § 2.2, causal inference involves four key steps, including structure learning, parameter learning, probabilistic inference, and counterfactual inference. We show how to combine the ZKP operators to implement these modules in zkCLEAR.

Structure Learning. Causal inference often requires a causal graph to represent the relationships between variables, which can be obtained from either human inputs or data-driven methods. To support versatile applications, zkCLEAR provide both options. The data-driven approach, also known as structure learning, aims to recover the underlying causal graph from observational data. Traditional structure learning methods suffer from either complex dynamic control flows and rules (e.g., PC algorithm [51]) or computationally expensive iterative search processes (e.g., GES algorithm [10]). Recently, supervised causal learning has emerged as a promising alternative which leverages the power of deep learning to induce causal structures from data [29, 37]. As a constant-time algorithm (i.e., only one forward pass is needed), it is much more efficient to implement in ZKP than traditional structure learning methods. We therefore employ the AVICI model [37], a pretrained DNN for causal discovery. Specifically, it takes the entire dataset as input and predicts the causal graph's adjacency matrix. Trained on a large corpus of data-graph pairs, AVICI makes relatively weak assumptions compared to traditional methods. In our framework, we treat AVICI as a black-box inference module and execute it within $C_{DNN\text{-}Inference}$. To ensure correctness, we then feed the predicted graph into the $C_{Acyclic\text{-}Check}$ circuit, which enforces that the output is a valid DAG before it enters subsequent causal-analysis modules.

Parameter Learning. When causal relationships are established, the next step is to learn the functional forms of the relationships from the causes to the effects, which is known as parameter learning. In accordance with the settings in most (non-private) causal inference frameworks, we assume that the functional form is linear and use our $C_{Linear\text{-}Training}$ circuit and $C_{Linear\text{-}Inference}$ circuit to learn the parameters of the linear functions and perform inference, respectively. We leave the extension to complex non-linear functions to future work, which could presumably be achieved by ZKP training of deep learning models.

Probabilistic Inference. Probabilistic inference aims to estimate the probability distribution of a target variable given the observed values of other variables. While standard machine learning techniques can be used for this task, in the context of causal inference, it is more principled to leverage the structure of the causal graph to guide the inference process. In zkCLEAR, we implement the belief propagation algorithm [43] to perform probabilistic inference. Belief propagation operates as a message-passing procedure on a factor graph derived from the causal model. Through an iterative process, the algorithm updates the belief (i.e., marginal distribution) at each node based on messages received from its neighbors. This process can be framed as a series of model inference tasks. Accordingly, we implement it using a series of $C_{DNN\text{-}Inference}$ circuits, where each node in the graph is associated with a predictive model trained to estimate its value based on its parents.

Counterfactual Inference. This is a cornerstone of causal reasoning, aiming to estimate potential outcomes under hypothetical interventions. Typically, it is achieved through do-calculus [4]. The do-calculus operator aims to compute the interventional distribution of a target variable X_{target} under an intervention on another variable $X_{intv} \leftarrow x_{intv}$. Hence, it starts by generating an interventional graph G^{intv} using the $C_{Interv.\text{-}Graph}$ circuit, which captures the structural changes induced by the intervention. Given this modified graph, the operator proceeds to estimate the distribution $P(X_{target}|do(X_{intv} = x_{intv}))$. Following the topological order of G^{intv}, each node X_i is assigned a counterfactual value $X'_i \leftarrow f_{X_i}(X'_{Pa(i)})$, where $f_{X_i}(\cdot)$ is the structural equation learned during the **Parameter Learning** phase. To reduce computational overhead, we prune the evaluation of X'_i whenever X_i is d-separated from X_{target} given X_{intv}, determined via the $C_{d\text{-}Separation}$ circuit.

This pruning is justified by conditional independence: if X_i is d-separated from X_{target}, it doesn't affect counterfactual outcome and can be omitted from computation. We can thus estimate the interventional distribution of X_{target} under the intervention with these operators.

5.2 Application

To illustrate the applications, we continue from Example 1.2.

Example 5.1. *Extending Example 1.2, suppose the healthcare authority asks: "What if patients in Region B had received the same treatment as those in Region A?" Using zkCLEAR, the provider answers this in a privacy-preserving and verifiable manner. The* ***Parameter Learning*** *module extracts causal relationships from committed data; the* ***Counterfactual Inference*** *module simulates outcomes under the new treatment assignment; and a ZK proof certifies the result without revealing patient information.*

To further showcase the practical applicability of zkCLEAR, we implement three representative applications of prescriptive analytics from the literature: (1) **Lewis** [19], a causal explanation method for explaining black-box models, (2) **Hyper** [18], a hypothetical reasoning algorithm, including both "what-if" and "how-to" analysis, and (3) **TACI** [26], a debugging algorithm of ML fairness. Due to limited space, we leave clarifying their real-world privacy concerns and presenting the core constructions of these applications under zkCLEAR in the appendix [1].

5.3 Proof Generation

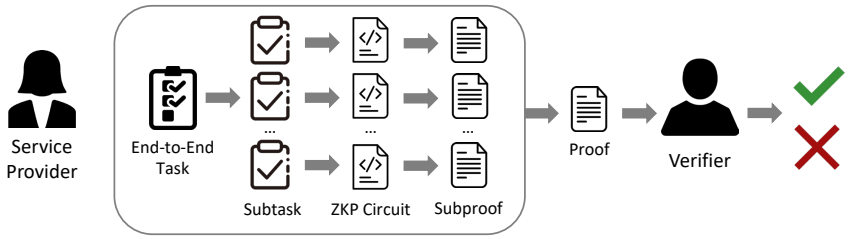


Fig. 7. Workflow decomposition and proof generation.

Building on the elementary ZKP operators and causal inference modules, we now demonstrate how to compose them into a complete causal prescriptive analytics workflow with efficient proof generation through workflow decomposition.

We represent the entire workflow as a computational DAG, where each node corresponds to a sub-task (e.g., d-separation, ATE estimation), and edges capture data dependencies. As illustrated in Fig. 7, rather than producing a single monolithic proof, we decompose the DAG into smaller subgraphs, each representing a self-contained computation unit. A separate proof is generated for each unit, which enables modular verification and eliminates the need to re-prove unchanged components.

For deep models using the $C_{DNN\text{-}Inference}$ circuit, we apply additional slicing at the model level that treats each neural network layer as an independent subgraph. This fine-grained partitioning enhances parallelism and improves scalability.

To ensure integrity across the workflow, we use cryptographic hashes to commit to the inputs and outputs of each subgraph. These commitments are validated at each computational boundary to maintain consistency throughout the DAG. Once all sub-proofs are verified, they are aggregated into a final proof certifying the correctness of the entire workflow.

6 Evaluation

We aim to answer the following evaluation questions (EQs):

- **EQ1: Capability.** How faithful and efficient is zkCLEAR in supporting real-world end-to-end workloads?
- **EQ2: Efficiency.** How does zkCLEAR perform under varying conditions compared to general-purpose ZKP systems?
- **EQ3: Ablation Study.** How do our optimization techniques contribute to the overall performance of zkCLEAR?

6.1 Experiment Setup

Datasets. For our case study evaluations, we use the following established datasets commonly employed in causal analysis research: (1) COMPAS [31], (2) Adult [5], (3) German [24], and (4) German-Syn [18, 19]. The first three are standard benchmarks widely studied in the literature, which we use without modification. For German-Syn, we follow the setup in [18, 19] and generate a synthetic dataset that mirrors the causal structure of the original German dataset. We generated 10,000 samples for this synthetic dataset, which were used to evaluate the XAI and hypothetical reasoning case studies. For low-level benchmarking, we employ randomly generated synthetic data in two forms—causal graphs and inference data—to evaluate different components of zkCLEAR based on their input requirements.

Baselines. Given the absence of specialized frameworks for ZKP causal inference, we compare zkCLEAR against two general-purpose ZKVM systems: RISC0 [57] and SP1 [30]. Both systems compile Rust programs into ZKP circuits. We implement the same algorithms in these systems to ensure a fair comparison.

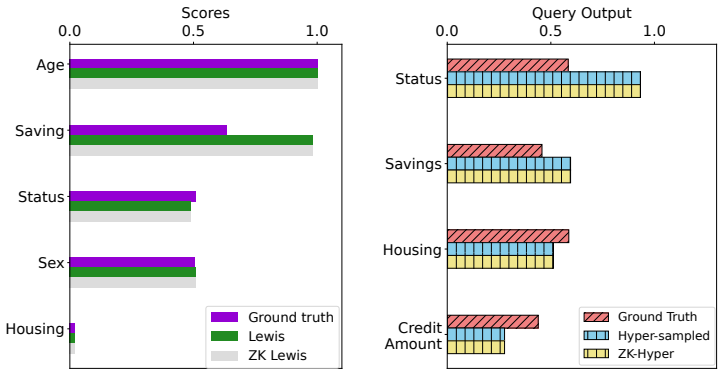
Metrics. We report total proof generation time which is the most concerning metric, verification time, and proof size. All result are averaged over five independent runs. For proof generation time, we further break it down into preprocessing and true proof generation phases, and compare them with corresponding non-ZKP implementations to identify bottlenecks. We also report the number of subproofs generated for these workloads. For faithfulness evaluation, we compare the outputs of zkCLEAR with the non-ZKP implementations of the algorithms to ensure result consistency.

Implementation. We implement zkCLEAR in Rust, with a total of 43701 lines of code. All experiments are conducted on a server with dual Intel(R) Xeon(R) Gold 6444Y CPUs @ 3.60GHz and 256GB RAM. We enabled multithreading across 64 threads and configured each group of 16 cores for parallel proof generation. We use the default configuration for all baselines.

6.2 EQ1: Capability

In this evaluation question, we assess the practical capability of zkCLEAR by applying it to three end-to-end, real-world causal applications described in § 5.2. Specifically, we evaluate whether zkCLEAR can faithfully replicate the analytical behavior of non-ZKP causal pipelines, while maintaining reasonable efficiency in terms of proof generation and verification overhead. Additionally, we provide detailed performance breakdowns to identify computational bottlenecks and offer a holistic assessment of the ZKP-induced overhead.

Lewis. Following the experiments in Lewis [19], we evaluate ZK-Lewis on the German-Syn dataset and compare its generated explanation scores against both a vanilla non-ZKP Lewis implementation and the ground truth values. As shown in Fig. 8a, the explanation scores generated by ZK-Lewis closely align with both plaintext estimates and ground truth values across different variables. The average deviation from the ground truth is within 12.1%. This consistency validates zkCLEAR's effectiveness in preserving the analytical integrity of causal explanation methods while providing



(a) ZK-Lewis (b) ZK-Hyper

Fig. 8. Results of ZK-Lewis and ZK-Hyper compared to non-ZKP implementations and ground truth.

Table 2. ZK-Lewis overhead.

Metric	Age	Sex	Saving	Status	Housing
Non-ZKP runtime (s)	0.622	0.644	4.258	4.465	4.269
ZKP preprocessing (s)	1.695	1.525	15.085	15.131	15.290
Proof construction (s)	10.312	9.453	97.479	90.625	99.870
Total proving (s)	12.007	10.978	112.564	105.756	115.160
Proof size (MB)	1.713	1.713	15.823	15.823	15.823
Number of subproofs	8	8	52	52	52
Verification (s)	0.363	0.358	3.280	3.324	3.176

robust ZKP guarantees. To evaluate performance, Table 2 details the runtime breakdown for each attribute. As expected, ZKP execution adds considerable overhead when compared to the plaintext baseline (0.622-4.465 seconds). Specifically, we observe that attributes like Saving, Status, and Housing incur the highest computational cost under ZKP, with total proving times ranging from 105.756 to 115.160 seconds, proof sizes reaching 15.823 MB, and verification times between 3.176 and 3.324 seconds. These attributes require 52 subproofs, reflecting a more complex causal structure with larger circuits. In contrast, simpler attributes like Age and Sex incur lower proving times (10.978–12.007s), smaller proofs (1.7 MB), under 0.4s verification, and only 8 subproofs. This reduction in overhead stems from their structural role as leaf nodes in the causal graph, which reduces the depth and complexity of both do-calculus reasoning and probabilistic inference steps. Proof construction dominates total proving time, while preprocessing contributes less. Overall, despite cryptographic overheads versus plaintext inference, proving and verification remain tractable for practical use.

Hyper. In line with [18], we evaluate ZK-Hyper's capability to answer both "what-if" and "how-to" queries using the German-Syn dataset. We report the results of the "what-if" queries in Fig. 8b. Overall, we observe that ZK-Hyper generally align well with its non-ZKP counterpart. However, we observe a slight deviation compared to the ground truth values. This is primarily due to the limited expressive power of the linear regression model used in our framework, which could be further improved by incorporating more complex models, such as random forests. However, despite this limitation, when used in "how-to" queries which prioritizes ranking over absolute values, ZK-Hyper correctly identifies account status and housing attributes as the optimal intervention, consistent with the baseline outcome. Table 3 shows the full breakdown: plaintext execution is fast

Table 3. ZK-Hyper overhead.

Metric	Status	Saving	Housing	Credit
Non-ZKP runtime (s)	1.165	1.089	1.087	1.092
ZKP preprocessing (s)	3.492	3.481	3.528	3.517
Proof construction (s)	21.593	20.733	21.690	21.485
Total proving (s)	25.085	24.214	25.218	25.002
Proof size (MB)	3.726	3.726	3.726	3.726
Number of subproofs	12	12	12	12
Verification (s)	0.782	0.777	0.785	0.793

Table 4. Results for ZKP fairness debugging. *C/A/G denote Compas, Adult, and German datasets, respectively.*

Metric	C_Sex	C_Race	A_Sex	A_Race	G_Age	G_Sex
Non-ZKP runtime (s)	30.74	30.74	40.97	39.80	31.39	27.34
ZKP preprocessing (s)	429.00	429.00	641.01	602.39	875.70	918.05
Proof construction (s)	7190.21	7190.21	9575.99	9690.29	11517.88	12958.99
Total proving (s)	8664.48	7619.21	10217.00	10292.68	12393.58	13877.05
Proof size (MB)	961.54	855.15	1276.85	1194.07	1562.03	1682.37
Number of subproofs	362	332	506	470	397	449
Verification (s)	190.79	166.29	252.99	233.91	307.16	322.17
F1 score	1.00	1.00	0.97	1.00	1.00	1.00

(1.087–1.165s), ZKP preprocessing takes ~3.5s, and proof construction ~21.5s. Total proving stays under 26s, verification under 0.8s, with 12 subproofs per query. These results demonstrate that zkCLEAR can support practical ZKP-based "what-if" reasoning with modest overhead. The results on "how-to" queries follows a similar trend and are proportional to "what-if" queries. We omit the details here for brevity, but they are available in the appendix [1].

Fairness Debugging. Finally, we follow the setup in TACI [26] and present the results for ZK-TACI on COMPAS, Adult, and German datasets. We report the results in Table 4. ZK-TACI incurs significantly higher proving costs than the other applications. While plaintext execution takes 27.34–40.97s, proving requires 7,619–13,877s, with verification at 166–322s and proof sizes of 855–1,682 MB. The high subproof count (332–556) reflects much greater circuit complexity. This higer overheads is expected, as it involves the *Structure Learning* module, which in turn invokes the $C_{DNN\text{-}Training}$ on a large pretrained Transformer model to analyze the causal structure of the dataset. Also, compared to the previous two applications, ZK-TACI handles a larger data and thereby a more complex causal graph (with 45 nodes and 135 edges on average). Also, in ATE calculations, we need to compute ATE values for all common ancestors of the trade-off attribute nodes, leading to hundreds of ATE computations. Despite this computational intensity, the overhead remains manageable, especially considering that ML fairness debugging is typically performed offline. Regarding accuracy, we use F1 score to measure how faithfully the ZKP variants replicate the outputs of their non-ZKP counterparts. As shown in the table, ZK-TACI achieves perfect macro-F1 scores (1.000) in five out of six cases. The slight reduction in performance for the "A_Sex" case is due to quantization errors that arise when handling floating-point numbers. This trade-off is made to optimize arithmetic operations within the ZKP framework.

Discussion on Deployment and Verifier Overhead. Verification remains modest for ZK-Lewis and ZK-Hyper (under 3.5s and 1s), but scales with proof size and subproof count. ZK-TACI, used for fairness debugging, can take minutes due to large proofs (up to 1.7 GB) and many subproofs (up to 506). In client-server settings, these costs can be amortized or parallelized. On blockchains,

however, smart contracts enforce sequential verification, limiting batch verification. To reduce latency, a "trust-but-verify" model can return results immediately and verify proofs asynchronously. On Polygon [45] (a ZKP-friendly blockchain platform), estimated verification costs are \$1.65–\$15.26 for ZK-Lewis, \$3.61 for ZK-Hyper, and \$378–\$719 for ZK-TACI. This suggests that while on-chain verification is viable for lightweight tasks, complex workloads may require off-chain verification, batched auditing, or proof recursion.

Takeaway for EQ1. Our evaluation across three diverse causal prescriptive analytics tasks, including XAI, hypothetical reasoning, and fairness debugging, demonstrates that zkCLEAR can faithfully support complete causal workflows and replicate non-ZKP outputs with strong ZKP guarantees. The performance breakdown reveals that proof construction is the dominant component of the total proving time, while preprocessing and verification contribute a smaller share. Despite the added overhead from ZKP computation, the resulting proving and verification costs remain tractable across tasks of varying complexity. These results highlight zkCLEAR's practical capability to serve as a foundation for privacy-preserving causal analytics in real-world applications.

6.3 EQ2: Efficiency

To quantify zkCLEAR's efficiency, we evaluate zkCLEAR's performance at both the operator and module levels, and assess how performance scales with input size. In addition, we present additional scalability evaluations in the appendix [1].

Comparison with ZKVM Baselines. Following our experimental setup, we compared zkCLEAR against the general-purpose ZKVM systems RISC0 and SP1. We tested performance on 10 input samples (data samples or causal graphs depending on the input format) and 10 attributes for the causal graph. The results are presented in Table 5 and Table 6.

For operator-level evaluation, zkCLEAR's operators consistently outperform the baselines in terms of proof generation time, verification time, and proof size. As shown in Table 5, zkCLEAR achieves significantly faster proving times, ranging from 6.5× to 25.6× faster than RISC0 and 10.7× to 35.1× faster than SP1 across different operators. Moreover, zkCLEAR generates proof sizes that are up to 29.0× smaller than RISC0 and 214.5× smaller than SP1. For verification, zkCLEAR demonstrates comparable or better performance. With the exception of the ancestor identification test—where our verifier is 1.7× slower (44 ms vs. 26 ms) but still under 50 ms—zkCLEAR verifies proofs 1.2× to 27.6× faster than RISC0 and 2.8× to 18.3× faster than SP1. The performance advantage is particularly pronounced for complex operations like d-separation testing, where zkCLEAR exhibits 25.6× faster proof generation than RISC0 and 11.1× faster than SP1.

For module-level evaluation (Table 6), zkCLEAR continues demonstrating superior performance where direct comparisons are possible. For parameter estimation, zkCLEAR achieves 3.3× faster proving times compared to RISC0 and 2.2× faster compared to SP1, while generating proofs that are 1.7× and 11.6× smaller, respectively. Similarly, for counterfactual inference, zkCLEAR proves 28.4× faster than RISC0 and 10.9× faster than SP1, with significantly smaller proof sizes. Since structure learning and probabilistic inference depend on DNN inference capabilities, such tooling on existing ZKVM baselines is not applicable, and we do not report the results.

Scalability. We evaluated zkCLEAR's scalability by assessing its performance under varying numbers of samples. Table 7 shows that zkCLEAR's causal primitives exhibit reasonable scalability with the number of samples. For instance, even the most computationally intensive operator, d-separation, has a proof generation time of 353.414 seconds at 1000 samples, which we consider a manageable overhead for many real-world applications.

Table 8 indicates that most of zkCLEAR's causal modules scale well with the number of samples (up to 1,000 attributes). Counterfactual inference, despite being the most computationally intensive among these well-scaling modules, still maintains reasonable performance with 1000 samples

Table 5. Runtime and communication overhead of causal operators: zkCLEAR vs. ZKVM baselines (Risc0 and SP1).

Protocol	Prover time (s)	Proof size (MB)	Verifier time (s)
Ancestor Identification			
zkCLEAR	1.102	0.182	0.044
Risc0	27.339	0.267	0.026
SP1	13.621	1.716	0.147
Acyclicity Check			
zkCLEAR	0.719	0.123	0.026
Risc0	8.399	0.243	0.034
SP1	7.658	1.716	0.147
D-Separation Test			
zkCLEAR	3.727	0.592	0.131
Risc0	95.448	1.056	0.162
SP1	41.199	4.517	0.366
Interventional Graph Generation			
zkCLEAR	0.542	0.008	0.019
Risc0	3.536	0.232	0.141
SP1	6.919	1.716	0.146
Linear Model Training			
zkCLEAR	1.887	0.239	0.045
Risc0	25.931	0.763	0.112
SP1	20.619	1.789	0.148
Linear Model Inference			
zkCLEAR	0.197	0.020	0.008
Risc0	4.658	0.432	0.221
SP1	6.919	1.716	0.146

Table 6. Runtime and communication overhead of causal modules: zkCLEAR vs. ZKVM baselines (Risc0 and SP1).

Protocol	Prover time (s)	Proof size (MB)	Verifier time (s)
Structure Learning			
zkCLEAR	1089.677	7.481	15.249
Risc0	NA	NA	NA
SP1	NA	NA	NA
Parameter Learning			
zkCLEAR	4.352	0.148	0.044
Risc0	14.175	0.255	0.046
SP1	9.751	1.716	0.162
Probabilistic Inference			
zkCLEAR	1.441	0.019	0.041
Risc0	NA	NA	NA
SP1	NA	NA	NA
Counterfactual Inference			
zkCLEAR	4.680	0.445	0.099
Risc0	132.971	1.056	0.106
SP1	51.147	4.517	0.46

(60.158 seconds for proof generation, 7.038 MB proof size, and 0.302 seconds for verification). However, structure learning presents a notable exception, exhibiting significantly steeper scaling characteristics. Its proof generation time increases dramatically from 218.253 seconds with 1 sample

Table 7. Runtime (sec) and communication overhead (MB) of causal operators with the different number of instances ($10^0, 10^1, 10^2, 10^3$) when the number of attributes is 10.

Primitive	10^0			10^1			10^2			10^3		
	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)
Ancestor Iden.	0.102	0.010	0.006	1.102	0.182	0.044	4.947	0.857	0.148	48.585	8.558	2.092
Acyclicity Check	0.245	0.021	0.012	0.719	0.123	0.026	9.986	1.809	0.368	104.228	18.069	6.586
D-Separation Test	0.470	0.062	0.020	3.773	0.592	0.136	35.474	5.894	1.387	353.414	58.916	19.632
Inter. Graph Gen.	0.097	0.011	0.006	0.542	0.081	0.019	5.634	0.793	0.189	46.247	8.093	1.970
Linear Model Training	1.860	0.236	0.042	1.887	0.239	0.045	2.119	0.278	0.075	4.657	0.659	0.114
Linear Model Inference	0.091	0.005	0.004	0.197	0.020	0.008	1.456	0.171	0.033	13.189	1.685	0.288

Table 8. Runtime (sec) and communication overhead (MB) of causal modules with the different number of instances ($10^0, 10^1, 10^2, 10^3$) when the number of attributes is 10.

Primitive	10^0			10^1			10^2			10^3		
	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)
Structure Learning	218.253	0.748	3.074	1089.677	7.481	15.249	5047.604	74.839	75.735	27161.349	748.912	311.337
Parameter Learning	3.175	0.142	0.041	4.352	0.148	0.044	4.405	0.198	0.051	14.395	0.702	0.138
Probabilistic Infer.	1.246	0.019	0.033	1.441	0.019	0.041	1.673	0.019	0.045	3.518	0.019	0.051
Counterfactual Infer.	4.047	0.384	0.088	4.680	0.445	0.099	9.868	1.045	0.187	60.158	7.038	0.302

to 27,161.349 seconds with 1,000 samples, and the proof size grows from 0.748 MB to 748.912 MB. This scalability limitation arises from the inherent complexity of the structure learning algorithm, which relies on EZKL for intricate model inference. While optimizing EZKL's performance is beyond the scope of this work, we anticipate that future advancements in ZKML inference systems will lead to improved scalability for the structure learning module.

Takeaway for EQ2. The experimental results robustly demonstrate that zkCLEAR's domain-specific architecture provides performance gains of several orders of magnitude over general-purpose ZKVMs across various causal primitives and applications, achieving faster proof generation and verification times, along with smaller proof sizes. The scalability analysis reveals that zkCLEAR generally maintains reasonable performance with increasing data size and complexity, although the structure learning module currently exhibits less favorable scaling behavior due to its reliance on complex ZKML inference. We expect future progress in ZKML technologies to mitigate this limitation.

6.4 EQ3: Ablation Study

To rigorously evaluate the contribution of our optimization techniques, we conducted an ablation study comparing the full zkCLEAR implementation against two ablated versions: zkCLEAR without constraint reduction (o1) and zkCLEAR without parallel proof generation (o2). Given that operator-level primitives form the fundamental building blocks of zkCLEAR and cannot be further parallelized at that granularity, we only assessed the impact of constraint reduction (o1) on operator performance. Conversely, for the structure learning and probabilistic inference modules, which leverage EZKL, we focused solely on the effect of parallel proof generation (o2). We excluded the linear model inference operator and intervention graph generation from this analysis due to their inherent simplicity, rendering them unaffected by our optimization strategies. The results of this ablation study are presented in Table 9 and Table 10.

The results consistently underscore the significant performance gains achieved through our optimization strategies. At the operator level, the implementation of constraint reduction (o1) yields substantial reductions, averaging 63.13% in proof size and 67.36% in proof generation time. The verification time also sees a considerable average reduction of 70.61%. Notably, for the d-separation

Table 9. Ablation study at operator-level.

Protocol	Prover time (s)	Proof size (MB)	Verifier time (s)
Ancestor Identification			
zkCLEAR	1.102	0.182	0.044
zkCLEAR w/o o1	3.765	0.694	0.141
Acyclicity Check			
zkCLEAR	0.719	0.123	0.026
zkCLEAR w/o o1	5.066	0.885	0.150
D-Separation Test			
zkCLEAR	3.773	0.592	0.136
zkCLEAR w/o o1	33.149	5.808	1.277
Linear Model Training			
zkCLEAR	1.887	0.239	0.045
zkCLEAR w/o o1	2.494	0.246	0.077

Table 10. Ablation study at module-level.

Protocol	Prover time (s)	Proof size (MB)	Verifier time (s)
Structure Learning			
zkCLEAR	1089.677	7.481	15.249
zkCLEAR w/o o1	NA	NA	NA
zkCLEAR w/o o2	4320.998	7.481	15.252
Parameter Learning			
zkCLEAR	4.352	0.148	0.044
zkCLEAR w/o o1	8.843	0.837	0.143
zkCLEAR w/o o2	18.964	0.148	0.051
Probabilistic Inference			
zkCLEAR	1.441	0.019	0.041
zkCLEAR w/o o1	NA	NA	NA
zkCLEAR w/o o2	5.963	0.019	0.043
Counterfactual Inference			
zkCLEAR	4.680	0.445	0.099
zkCLEAR w/o o1	54.839	6.400	1.534
zkCLEAR w/o o2	19.693	0.445	0.109

test, the benefits are even more pronounced, with proof generation and verification times decreasing by 88.62% and 89.35%, respectively.

At the module level, our optimizations show a large reduction in prover time across various modules. For instance, in the parameter learning module, disabling optimization o1 increases the prover time from 4.352s to 8.843s (a 103% increase), and the removal of o2 also leads to a 335% increase (18.964s). Notice that the parallel proof generation (o2) cannot optimize the proof size and verification time, as it works by enabling multiple proof generations in parallel. Similarly, in probability inference, the prover time rises from 1.441s to 5.963s without o2, indicating a 314% increase. The most dramatic improvements are observed in counterfactual inference, where disabling o1 leads to a prover time of 54.839s compared to 4.680s with full optimization—a 1072% increase. These results underscore the critical role of our proposed optimizations in achieving practical performance for complex inference and learning tasks within the zkCLEAR protocol framework.

Takeaway for EQ3. Our ablation study demonstrates that our optimization techniques are critical to achieving practical performance. Constraint reduction consistently improves proof efficiency across all operators, leading to an order-of-magnitude reduction in proving time and size. Parallel

proof generation further enhances scalability, especially for complex modules. This dual approach—first optimizing the work within each proof, then parallelizing the entire workload—is indispensable for making complex, end-to-end ZKP causal inference feasible for real-world applications.

7 Related Work

Privacy-Enhancing Techniques for Data Analytics. Prior research has explored privacy-preserving computation in various data analytics tasks. Secure multi-party computation (MPC), homomorphic encryption (HE), oblivious RAM (ORAM), and differential privacy (DP) have been applied to enable privacy-preserving query processing [12, 14, 38, 46]. While these approaches ensure privacy, they lack *verifiability*, i.e., the ability for external parties to verify results without accessing raw data. ZKPs fill this gap by enabling verifiable computation, which we adopt to enable privacy-preserving and verifiable prescriptive analytics.

Existing ZKP Systems. Table 1 summarizes the capabilities of representative ZKP systems. *Constraint-based frameworks* (e.g., Circom [6], Halo2 [11]) compile to arithmetic circuits with fine-grained control, but lack efficient support for real-valued operations, matrices, dynamic control flow, and structured data—making graph-based operations costly to encode. *ZK virtual machines* [30, 57] offer general-purpose programmability via CPU trace verification but incur high overhead on compute-heavy tasks. *ZKML frameworks* [25, 36, 52] efficiently support DNN inference, with some extending to verifiable training, but lack primitives for causal inference tasks such as ancestor queries and d-separation. zkCLEAR addresses these gaps with low-level ZKP operators and an integrated framework optimized for prescriptive analytics.

Causality in Database. Causality has been used in various database applications, including data integration [56], data cleaning [44, 50], query explanation [39, 41, 42, 48, 55], hypothetical reasoning [18] and system diagnosis [26, 40]. We implement three of these representative applications using zkCLEAR and demonstrate its effectiveness and adaptability.

8 Limitations and Future Work

In this section, we discuss these limitations and outline potential future directions.

Extension to Non-Linear Models. zkCLEAR currently supports linear causal models, which limits its applicability to scenarios where the relationships between variables are non-linear. With the advancements in ZKML technologies, we anticipate that future work will enable complex non-linear models, such as DNNs, to be integrated into zkCLEAR.

Extension to Relational Data. zkCLEAR focuses on single-table inputs to optimize for causal inference tasks. Extension to relational data is feasible via integration with ZK-SQL frameworks [35], which could enable prescriptive analytics over relational sources.

9 Conclusion

We have presented zkCLEAR, a ZKP-based framework for privacy-preserving and verifiable causal analytics. We support key components of causal inference, and design optimizations to reduce proof size and computational overhead. We show the utility of our framework through real-world applications and also evaluate its efficiency and scalability. We believe our work opens up new avenues for privacy-preserving causal inference and prescriptive analytics, and can be applied to various important domains of data science.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments. The HKUST authors were supported in part by a research fund provided by HSBC and the Technology Start-up Support Scheme for Universities (TSSSU) from Hong Kong ITC.

References

[1] 2025. Extended version of the paper with appendix. https://github.com/wangzhaoyu07/zkclear/extended-version.pdf.

[2] 2025. Research artifact. https://github.com/wangzhaoyu07/zkclear.

[3] Axiom. 2025. halo2-lib. https://github.com/axiom-crypto/halo2-lib.

[4] Alexander Balke and Judea Pearl. 2022. Probabilistic evaluation of counterfactual queries. In *Probabilistic and causal inference: The works of Judea Pearl*. 237–254.

[5] Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository.

[6] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. 2022. Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing* 20, 6 (2022), 4733–4751.

[7] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. 2017. The hunting of the SNARK. *Journal of Cryptology* 30, 4 (2017), 989–1066.

[8] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 560–574.

[9] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. 2016. Double/debiased machine learning for treatment and causal parameters. *arXiv preprint arXiv:1608.00060* (2016).

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

[11] Electric Coin Co. 2025. halo2. https://github.com/zcash/halo2.

[12] Wei Dong, Zijun Chen, Qiyao Luo, Elaine Shi, and Ke Yi. 2024. Continual observation of joins under differential privacy. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[13] Clemens Dubslaff, Kallistos Weis, Christel Baier, and Sven Apel. 2022. Causality in configurable software systems. In *Proceedings of the 44th International Conference on Software Engineering*. 325–337.

[14] Saba Eskandarian and Matei Zaharia. [n. d.]. ObliDB: Oblivious Query Processing for Secure Databases. *Proceedings of the VLDB Endowment* 13, 2 ([n. d.]).

[15] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. *Cryptology ePrint Archive* (2021).

[16] Davide Frazzetto, Thomas Dyhre Nielsen, Torben Bach Pedersen, and Laurynas Šikšnys. 2019. Prescriptive analytics: a survey of emerging trends and technologies. *The VLDB Journal* 28, 4 (2019), 575–595.

[17] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive* (2019).

[18] Sainyam Galhotra, Amir Gilad, Sudeepa Roy, and Babak Salimi. 2022. Hyper: Hypothetical reasoning with what-if and how-to queries using a probabilistic causal approach. In *Proceedings of the 2022 International Conference on Management of Data*. 1598–1611.

[19] Sainyam Galhotra, Romila Pradhan, and Babak Salimi. 2021. Explaining black-box algorithms using probabilistic contrastive counterfactuals. In *Proceedings of the 2021 International Conference on Management of Data*. 577–590.

[20] S GOLDWASSER, S MICALI, and C RACKOFF. 1989. The knowledge complexity of interactive proof systems. *SIAM journal on computing (Print)* 18, 1 (1989), 186–208.

[21] Binbin Gu, Juncheng Fang, and Faisal Nawab. 2024. PoneglyphDB: Efficient Non-interactive Zero-Knowledge Proofs for Arbitrary SQL-Query Verification. *arXiv preprint arXiv:2411.15031* (2024).

[22] Peter J Haas, Paul P Maglio, Patricia G Selinger, and Wang-Chiew Tan. 2011. Data is dead... without what-if models. *Proceedings of the VLDB Endowment* 4, 12 (2011), 1486–1489.

[23] Qijian He, Wei Yang, Bingren Chen, Yangyang Geng, and Liusheng Huang. 2020. Transnet: Training privacy-preserving neural network over transformed layer. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1849–1862.

[24] Hans Hofmann. 1994. Statlog (German Credit Data). UCI Machine Learning Repository.

[25] ZKonduit Inc. 2025. EZKL. https://ezkl.xyz/.

[26] Zhenlan Ji, Pingchuan Ma, Shuai Wang, and Yanhui Li. 2023. Causality-aided trade-off analysis for machine learning fairness. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 371–383.
[27] Kunming Jiang, Fraser Brown, and Riad S Wahby. 2025. CoBBL: Dynamic Constraint Generation for SNARKs. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3347–3363.
[28] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*. Springer, 177–194.
[29] Nan Rosemary Ke, Silvia Chiappa, Jane X Wang, Jorg Bornschein, Anirudh Goyal, Melanie Rey, Theophane Weber, Matthew Botvinick, Michael Curtis Mozer, and Danilo Jimenez Rezende. [n. d.]. Learning to Induce Causal Structure. In *International Conference on Learning Representations*.
[30] Succinct Labs. 2025. SP1. https://github.com/succinctlabs/sp1.
[31] Jeff Larson, Surya Mattu, Lauren Kirchner, and Julia Angwin. 2016. The compas dataset. https://github.com/propublica/compas-analysis.
[32] Steffen L Lauritzen. 1996. *Graphical models*. Vol. 17. Clarendon Press.
[33] Benton Li, Nativ Levy, Brit Youngmann, Sainyam Galhotra, and Sudeepa Roy. 2025. Fair and Actionable Causal Prescription Ruleset. *arXiv preprint arXiv:2502.19846* (2025).
[34] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. 2022. Federated learning on non-iid data silos: An experimental study. In *2022 IEEE 38th international conference on data engineering (ICDE)*. IEEE, 965–978.
[35] Xiling Li, Chenkai Weng, Yongxin Xu, Xiao Wang, and Jennie Rogers. 2023. Zksql: Verifiable and efficient query evaluation with zero-knowledge proofs. *Proceedings of the VLDB Endowment* 16, 8 (2023).
[36] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.
[37] Lars Lorch, Scott Sussex, Jonas Rothfuss, Andreas Krause, and Bernhard Schölkopf. 2022. Amortized inference for causal structure learning. *Advances in Neural Information Processing Systems* 35 (2022), 13104–13118.
[38] Qiyao Luo, Yilei Wang, Ke Yi, Sheng Wang, and Feifei Li. 2023. Secure sampling for approximate multi-party query processing. *Proceedings of the ACM on Management of Data* 1, 3 (2023), 1–27.
[39] Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. Xinsight: explainable data analysis through the lens of causality. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.
[40] Markos Markakis, An Bo Chen, Brit Youngmann, Trinity Gao, Ziyu Zhang, Rana Shahout, Peter Baile Chen, Chunwei Liu, Ibrahim Sabek, and Michael Cafarella. 2024. Sawmill: From Logs to Causal Diagnosis of Large Systems. In *Companion of the 2024 International Conference on Management of Data*. 444–447.
[41] Alexandra Meliou, Wolfgang Gatterbauer, Joseph Y Halpern, Christoph Koch, Katherine F Moore, and Dan Suciu. 2010. Causality in databases. *IEEE Data Engineering Bulletin* 33, 3 (2010), 59–67.
[42] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F Moore, and Dan Suciu. 2010. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *Proceedings of the VLDB Endowment* 4, 1 (2010).
[43] Judea Pearl. 2022. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Probabilistic and causal inference: the works of Judea Pearl*. 129–138.
[44] Alireza Pirhadi, Mohammad Hossein Moslemi, Alexander Cloninger, Mostafa Milani, and Babak Salimi. 2024. Otclean: Data cleaning for conditional independence violations using optimal transport. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–26.
[45] Polygon. 2025. Polygon zkEVM doc. https://docs.polygon.technology/zkEVM/
[46] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the twenty-third ACM symposium on operating systems principles*. 85–100.
[47] Jennie Rogers, Elizabeth Adetoro, Johes Bater, Talia Canter, Dong Fu, Andrew Hamilton, Amro Hassan, Ashley Martinez, Erick Michalski, Vesna Mitrovic, et al. 2022. VaultDB: A real-world pilot of secure multi-party computation within a clinical research network. *arXiv preprint arXiv:2203.00146* (2022).

[48] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1579–1590.
[49] Babak Salimi, Harsh Parikh, Moe Kayali, Lise Getoor, Sudeepa Roy, and Dan Suciu. 2020. Causal relational learning. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 241–256.
[50] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. 2019. Interventional fairness: Causal database repair for algorithmic fairness. In *Proceedings of the 2019 International Conference on Management of Data*. 793–810.
[51] Peter Spirtes, Clark N Glymour, and Richard Scheines. 2000. *Causation, prediction, and search*. MIT press.
[52] Suppakit Waiwitlikhit, Ion Stoica, Yi Sun, Tatsunori Hashimoto, and Daniel Kang. 2024. Trustless audits without revealing data or models. In *Proceedings of the 41st International Conference on Machine Learning*. 49808–49821.
[53] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*. 501–518.
[54] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy preserving vertical federated learning for tree-based models. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2090–2103.
[55] Brit Youngmann, Michael Cafarella, Amir Gilad, and Sudeepa Roy. 2024. Summarized Causal Explanations For Aggregate Views. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–27.
[56] Brit Youngmann, Michael Cafarella, Babak Salimi, and Anna Zeng. 2023. Causal Data Integration. *Proceedings of the VLDB Endowment* 16, 10 (2023), 2659–2665.
[57] RISC Zero. 2025. RISC Zero. https://github.com/risc0/risc0.
[58] Yizheng Zhu, Yuncheng Wu, Zhaojing Luo, Beng Chin Ooi, and Xiaokui Xiao. 2023. Secure and verifiable data collaboration with low-cost zero-knowledge proofs. *arXiv preprint arXiv:2311.15310* (2023).

Appendix

A Application

To demonstrate the practical applicability of zkCLEAR, we implement three representative applications of prescriptive analytics from the literature: (1) explaining black-box models [19], (2) hypothetical reasoning [18], and (3) debugging ML fairness [26]. In the following, we present the core constructions of these applications under zkCLEAR.

A.1 Explaining Black-Box Models. Modern machine learning models are increasingly deployed in high-stakes settings, yet their opaque decision-making raises concerns around interpretability. Our first application targets this challenge by supporting explainable AI (XAI) through causal analysis. Specifically, we implement Lewis [19] under zkCLEAR, which is an approach that quantifies the influence of input features on black-box model outcomes via counterfactual reasoning.

Technically, Lewis defines three causal explanation scores for a feature X under context $\mathbf{k}$: *Necessity (Nec)* assesses whether X is indispensable for a given outcome; *Sufficiency (Suf)* evaluates whether altering X alone could produce the outcome; and *Necessity and Sufficiency (NecSuf)* provides a joint characterization of both effects. These scores are formally defined as:

$$\mathrm{Nec}_x(\mathbf{k}) = \frac{\left(\sum_{c\in C} \Pr(o' \mid \mathbf{c}, \mathrm{do}(\mathbf{x}'), \mathbf{k})\ \Pr(\mathbf{c} \mid \mathbf{x}, \mathbf{k})\right) - \Pr(o' \mid \mathbf{x}, \mathbf{k})}{\Pr(o \mid \mathbf{x}, \mathbf{k})}$$

$$\mathrm{Suf}_x(\mathbf{k}) = \frac{\left(\sum_{\mathbf{c}\in C}\ \Pr(o \mid \mathbf{c}, \mathbf{x}, \mathbf{k}) \Pr(\mathbf{c} \mid \mathrm{do}(\mathbf{x}'), \mathbf{k})\right) - \Pr(o \mid \mathrm{do}(\mathbf{x}'), \mathbf{k})}{\Pr(o' \mid \mathrm{do}(\mathbf{x}'), \mathbf{k})}$$

$$\mathrm{NecSuf}_x(\mathbf{k}) = \sum_{\mathbf{c}\in C} \left(\Pr(o \mid \mathbf{x}, \mathbf{k}, \mathbf{c}) - \Pr(o \mid \mathrm{do}(\mathbf{x}'), \mathbf{c}, \mathbf{k})\right)\ \Pr(\mathbf{c} \mid \mathbf{k})$$

Each score depends on evaluating conditional and interventional probabilities that can be ideally computed using zkCLEAR's built-in modules. In particular, ① the *Parameter Learning* module estimates the structural equations needed to compute $\Pr(\cdot)$; ② the *Probabilistic Inference* module supports marginal and conditional queries over the causal model; and ③ the *Counterfactual Inference* module enables evaluation of $\mathrm{do}(\cdot)$ interventions. By composing these modules, zkCLEAR enables a transparent, privacy-preserving, and verifiable implementation of Lewis. This could further enhance the transparency and trustworthiness of black-box models, as users can independently verify the explanation scores without needing to access the underlying proprietary model.

A.2 Hypothetical Reasoning. Hypothetical reasoning, including both "what-if" and "how-to" analysis, is fundamental for strategic decision-making and risk assessment. It allows users to explore alternative scenarios and assess potential outcomes without enacting real-world changes, making it a core capability in prescriptive analytics.

To demonstrate zkCLEAR's effectiveness in this setting, we implement the Hyper algorithm [18], which operates over a causal model capturing the system's causal relationships. With zkCLEAR and user-provided causal graph, the quantitative causal relationships are learned from observational data using the *Parameter Learning* module. Once constructed, Hyper conducts "what-if" and "how-to" analysis. For "what-if" queries, it estimates the expected outcomes of an intervention using do-calculus

$$\textit{What-If} = E(\mathbf{Y} \mid \mathrm{do}(\mathbf{x})) \tag{4}$$

It quantifies the effect of setting treatment variable(s) to specific values $\mathbf{x}$. For "how-to" queries, it identifies the intervention $\mathbf{x}$ that optimizes a desired objective. For instance,

$$\textit{How-To} = \begin{cases} \arg\max_{\mathbf{x}} E(\mathbf{Y} \mid \mathrm{do}(\mathbf{x})) & \text{if maximizing} \\ \arg\min_{\mathbf{x}} E(\mathbf{Y} \mid \mathrm{do}(\mathbf{x})) & \text{if minimizing} \end{cases} \tag{5}$$

Hyper answers these queries using do-calculus to derive interventional outcomes and handles "how-to" queries by exhaustively evaluating intervention strategies over the learned SEM. These computations map directly to zkCLEAR's capabilities: the Counterfactual Inference module implements the do-calculus to answer interventional queries, while repeated calls to this module support optimization over candidate interventions. In this way, users can trust the predicted outcomes and the recommended interventions without needing to access sensitive data.

A.3 Debugging ML Fairness. As concerns over bias in machine learning systems grow, so does the need for tools that can help understand and mitigate fairness-related trade-offs. These trade-offs often arise between fairness and other performance metrics, such as accuracy or efficiency. To demonstrate how zkCLEAR can support this analysis, we implement the TACI (Trade-off Analysis using Causal Inference) algorithm [26].

TACI consists of two main components: (1) causal graph construction and (2) trade-off analysis. The causal structure is first learned from observational data using zkCLEAR's built-in structure learning modules. Once the causal graph is established, TACI performs trade-off analysis by identifying causal relationships between metrics and estimating their causal impact using *average treatment effect* (ATE).

To estimate ATE, zkCLEAR adopts the *double machine learning* (DML) [9]. DML proceeds in two stages: first, we train predictive models for the treatment variable T and the outcome variable Y using covariates X, producing fitted values $\hat{T} = f_T(X)$ and $\hat{Y} = f_Y(X)$. We then compute residuals $\tilde{T} = T - \hat{T}$ and $\tilde{Y} = Y - \hat{Y}$ to remove confounding effects. In the second stage, we regress $\tilde{Y}$ on $\tilde{T}$, estimating the ATE as $\frac{\sum_{i=1}^{n} \tilde{T}_i \tilde{Y}_i}{\sum_{i=1}^{n} \tilde{T}_i^2}$. Since both stages reduce to linear regression training and inference, they are implemented in zkCLEAR using the $C_{Linear\text{-}Training}$ and $C_{Linear\text{-}Inference}$ circuits, along with basic arithmetic operators. Combined with ancestor identification, this enables zkCLEAR to fully support the TACI algorithm for systematically debugging fairness trade-offs in machine learning workflows.

B How-to Query Results in RQ1

To address the "how-to" query, aligning with the original work, we considered a query aiming to maximize the fraction of individuals receiving good credit. ZK-Hyper identified updating the account status and housing attributes as the optimal intervention, a finding consistent with the original work. The proof generation and verification times for this "how-to" query were 311.182 seconds and 13.576 seconds, respectively, with a proof size of 17.882 MB. As expected, these overhead metrics are greater than those for the "what-if" queries, as the "how-to" query necessitates multiple iterations of the "what-if" analysis to identify the optimal solution. However, the increase in overhead is not excessively large because the most computationally intensive step, parameter learning for the SEM, is performed only once for both types of queries. Consequently, even with numerous underlying "what-if" computations, the overall overhead for the "how-to" query remains within an acceptable range. These results further indicate that zkCLEAR can effectively support ZKP-based hypothetical reasoning for more complex query types.

C Full Results of the Scalability Evaluation in RQ2

Received April 2025; revised July 2025; accepted August 2025

Table 11. Runtime (sec) and communication overhead (MB) of causal operators with the different number of instances ($10^0, 10^1, 10^2, 10^3$) when the number of attributes is 10.

Primitive	10^0			10^1			10^2			10^3		
	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)
Ancestor Iden.	0.102	0.010	0.006	1.102	0.182	0.044	4.947	0.857	0.148	48.585	8.558	2.092
Acyclicity Check	0.245	0.021	0.012	0.719	0.123	0.026	9.986	1.809	0.368	104.228	18.069	6.586
D-Separation Test	0.470	0.062	0.020	3.773	0.592	0.136	35.474	5.894	1.387	353.414	58.916	19.632
Inter. Graph Gen.	0.097	0.011	0.006	0.542	0.081	0.019	5.634	0.793	0.189	46.247	8.093	1.970
Linear Model Training	1.860	0.236	0.042	1.887	0.239	0.045	2.119	0.278	0.075	4.657	0.659	0.114
Linear Model Inference	0.091	0.005	0.004	0.197	0.020	0.008	1.456	0.171	0.033	13.189	1.685	0.288

Table 12. Runtime (sec) and communication (MB) overhead of causal primitives with the different number of attributes (5, 10, 15, 20) when the number of instances is 10.

Primitive	5			10			15			20		
	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)
Ancestor Iden.	0.339	0.025	0.014	1.102	0.182	0.044	2.453	0.191	0.055	3.407	0.334	0.085
Acyclicity Check	0.415	0.049	0.017	0.719	0.123	0.026	2.281	0.403	0.096	4.274	0.711	0.150
D-Separation Test	0.827	0.108	0.037	3.773	0.592	0.136	11.180	1.794	0.374	24.938	4.056	0.850
Inter. Graph Gen.	0.349	0.023	0.014	0.542	0.081	0.019	1.110	0.184	0.052	2.325	0.324	0.079
Linear Model Training	1.294	0.086	0.034	1.887	0.239	0.045	7.147	0.469	0.139	7.387	0.773	0.212
Linear Model Inference	0.357	0.019	0.010	0.197	0.020	0.008	0.414	0.021	0.014	0.475	0.022	0.016

Table 13. Runtime (sec) and communication overhead (MB) of causal modules with the different number of instances ($10^0, 10^1, 10^2, 10^3$) when the number of attributes is 10.

Primitive	10^0			10^1			10^2			10^3		
	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)
Structure Learning	218.253	0.748	3.074	1089.677	7.481	15.249	5047.604	74.839	75.735	27161.349	748.912	311.337
Parameter Learning	3.175	0.142	0.041	4.352	0.148	0.044	4.405	0.198	0.051	14.395	0.702	0.138
Probabilistic Inference	1.246	0.019	0.033	1.441	0.019	0.041	1.673	0.019	0.045	3.518	0.019	0.051
Counterfactual Inference	4.047	0.384	0.088	4.680	0.445	0.099	9.868	1.045	0.187	60.158	7.038	0.302

Table 14. Runtime (sec) and communication (MB) overhead of causal modules with the different number of attributes (5, 10, 15, 20) when the number of instances is 10.

Primitive	5			10			15			20		
	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)	P t (s)	P s (MB)	V t (s)
Structure Learning	463.497	3.781	3.402	1089.677	7.481	15.249	1643.987	10.870	35.237	2126.427	13.948	63.366
Parameter Learning	0.130	0.002	0.004	4.352	0.148	0.044	8.098	0.276	0.057	15.233	0.552	0.113
Probability Inference	1.345	0.017	0.036	1.441	0.019	0.041	1.517	0.021	0.042	1.563	0.022	0.045
Counterfactual Inference	0.083	0.002	0.003	4.680	0.445	0.099	15.304	2.026	0.452	43.417	6.440	0.701