big data and cognitive computing

MDPI

Article

Declarative Causal Inference and Counterfactual Reasoning via SQL-Dialect Operators

Ronnit Peter [1], Suprio Ray [1,*] and Moulay A. Akhloufi [2]

[1] Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada; ronnit.peter@unb.ca
[2] Department of Computer Science, Université de Moncton, Moncton, NB E1A 3E9, Canada; moulay.akhloufi@umoncton.ca
* Correspondence: sray@unb.ca

Abstract

Relational databases power high-stakes decisions in lending, healthcare, and justice, yet SQL lacks native constructs for causal and counterfactual reasoning. Prior SQL-based causal systems address parts of this gap but do not unify treatment-effect estimation with counterfactual generation in a single, composable SQL surface. We present a system that extends the SQL dialect with two declarative operators: `EXPLAIN_CAUSALLY_WHY` (ψ) for estimating average and conditional treatment effects via meta-learners, and `EXPLAIN_COUNTERFACTUAL` (φ) for generating diverse, constraint-respecting alternatives via a hybrid KD-tree/LSH pipeline. Both operators consume standard SQL relations (joins, filters, projections) and return table-valued results with optional diagnostics, confidence intervals, and feasibility metrics. We formalize the operators in relational algebra, and describe our prototype system called PsiQL. On four evaluation datasets, PsiQL recovers a protective TWINS treatment effect, returns a non-significant COMPAS point ATE with imbalance diagnostics, flags HMDA covariate imbalance via built-in SMD checks, and generates constraint-respecting counterfactuals; a synthetic Census run serves as a balanced pipeline proof-of-concept alongside a real ACS diagnostic under severe imbalance.

Keywords: declarative causal inference; SQL-dialect operators; counterfactual reasoning; meta-learners; KD-tree; locality-sensitive hashing; algorithmic recourse; fairness-aware machine learning; relational algebra; in-database analytics

1. Introduction

SQL was designed to let non-programmers ask questions of data [1]. It succeeded: relational databases now power high-stakes decisions in lending, healthcare, and criminal justice. However, the questions people actually need answered have outgrown what SQL can express. "Does race/gender affect loan approval after controlling for creditworthiness?" is a causal question. "What minimal change to a denied application would flip it to approval?" is a counterfactual question. SQL has no native constructs for either. The current workaround is to export data to Python 3.10 or R 4.3+ , fit causal models externally, and re-import, which fractures provenance, bypasses query plan optimization, and breaks the audit trail that regulations like GDPR Article 22 require [2]. Post hoc methods like SHAP [3] and LIME [4] quantify feature importance but do not answer interventional questions [5,6]; as Lipton [7] puts it, interpretability remains ill-defined without causal grounding.

Check for updates

Academic Editor: Fabrizio Messina

Received: 9 June 2026
Revised: 17 July 2026
Accepted: 4 August 2026
Published: 17 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

Existing approaches leave a clear gap. External causal toolkits (CausalML [8], EconML [9], DoWhy [10]) provide meta-learners and diagnostics but require data export from the database. ZaliQL [11,12] and CaRL [13] bring causal inference closer to the database (ZaliQL via SQL-native propensity methods, CaRL via Datalog-like causal rules), but neither supports meta-learners, heterogeneous effects, or counterfactual generation. In-database ML systems (MADlib [14], BigQuery ML) support prediction but not causal identification or counterfactual feasibility. To our knowledge, no existing system combines treatment-effect estimation with constraint-respecting counterfactual generation in a composable SQL-dialect surface.

Technical difficulties are not merely syntactic. Embedding causal inference into SQL is harder than it first appears, and the difficulty is not syntactic. Identification assumptions, such as unconfoundedness, positivity, and stable unit treatment value assumption (SUTVA) [15], are not expressible in the relational algebra, since they require domain knowledge that the query planner cannot enforce. As model training is iterative and stateful, meta-learners require cross-fitting, bootstrapping, and multi-pass data access that violate the set-at-a-time model of relational operators. Feasibility constraints for counterfactuals add another layer of complexity because immutability, permitted ranges, and monotonicity interact with the learned model in ways that require custom constraint propagation beyond a simple predicate pushdown. Our proposed approach places the causal execution engine below the SQL binder and above the database engine, giving it full access to the relational plan while delegating model training and constraint enforcement to purpose-built Python 3.10 components.

There is also a practical case for keeping the analysis in SQL. The practical argument for keeping causal analysis inside SQL is data provenance: the FROM/JOIN/WHERE clauses define exactly which records contributed to an estimate, and that definition is auditable. There is also an optimization argument, since predicate pushdowns and LIMIT clauses reduce the analysis population before expensive training. A composability argument applies as well, which is as follows. The ATE (Equation (1)) is returned as a relation that can be joined with a policy table, filtered by a fairness threshold, or aggregated across jurisdictions without custom glue code. Whether these benefits outweigh the engineering cost of a dialect extension depends on the deployment context; for exploratory fairness analysis in regulated domains, we believe they do.

Building on prior work [16], we present a framework, PsiQL, that extends the SQL dialect with two declarative operators: `EXPLAIN_CAUSALLY_WHY` (ψ) for treatment-effect estimation via meta-learners [17] (S/T/X/R/DR-learners, uplift, ensembles) and `EXPLAIN_COUNTERFACTUAL` (φ) for generating diverse, constraint-respecting alternatives via a hybrid KD-tree/LSH indexing pipeline. Both operators consume relations defined by standard SQL (FROM/JOIN/WHERE); optional `GROUP BY` yields per-stratum effects under ψ (Rule 4, Section 4). and does not yield per-group CF sets under φ. Both return relations with optional confidence intervals, diagnostics, and feasibility metrics. A running example (Section 3) illustrates the full I/O trace for a fair-lending scenario based on the HMDA dataset [18].

Our main contribution is a systems one: PsiQL offers a unified SQL-native system that (a) eliminates boilerplate and export/import steps, (b) supports automatic identification strategy selection (meta-learners, ensembles, diagnostics), (c) integrates causal estimation and counterfactual generation in a single query, and (d) applies PCA-based dimensionality reduction when the covariate dimension exceeds 15 so that KD-tree indexing remains effective for high-dimensional counterfactual search. The PCA auto-enable is a heuristic specific to SQL-embedded φ that addresses the degradation of KD-trees in $d \gtrsim 20$ dimensions.

Relative to prior approaches, the underlying algorithms—meta-learners, DiCE-style counterfactual search, classical PSM/IPW, and KD-tree/LSH indexing—are established; the contribution is the systems packaging: unified ψ+φ SQL-dialect operators with table-valued, composable outputs and CQP binding that maps relational plans to causal specifications. The integration contribution is an end-to-end in-database pipeline that combines automatic ensemble selection, diagnostics (e.g., SMD), and constraint-respecting counterfactual generation in one query surface. Relative to prior work, ZaliQL and CaRL bring causality closer to the database but lack meta-learners, heterogeneous effects, and counterfactual generation; CausalML/DoWhy provide rich estimators outside the database; and in-DB ML systems support prediction but not causal identification or feasible counterfactuals.

We evaluate PsiQL with four datasets, HMDA, COMPAS, TWINS and Census, (see Section 7). Section 7 evaluates the system on HMDA, COMPAS, TWINS, and Census: TWINS recovers a protective birth-weight effect; COMPAS yields a non-significant point ATE under covariate imbalance; HMDA is flagged by SMD diagnostics; and counterfactual quality is reported for COMPAS and TWINS at $N = 100$.

The contributions of this paper are as follows.

- We define two SQL-dialect causal operators that accept standard SQL subqueries and return table-valued results: `EXPLAIN_CAUSALLY_WHY` (ψ) estimates ATE/CATE, and `EXPLAIN_COUNTERFACTUAL` (φ) generates feasible counterfactuals.
- A binder maps relational query plans (WHERE, JOIN, projections) to causal specifications with predicate pushdown and schema validation; optional `GROUP BY` is scheduled per Rule 4 (Section 4). We formalize this as a causal query plan (CQP) in Section 5.
- The causal engine integrates established meta-learners (S/T/X/R/DR), ensemble estimation, cross-fitting, bootstrap CIs, and SMD diagnostics rather than introducing new estimators.
- The counterfactual engine uses a dual KD-Tree + MinHash LSH index (configurable via `USE_KDETREE`/`USE_LSH`) to seed DiCE with training instances, enforcing immutability, range, and monotonicity constraints. The default enables both indices as a diversity–proximity trade-off (Section 7.4); KD-tree-only mode favors proximity but can return near-duplicate neighbors.
- We evaluate on TWINS, COMPAS, HMDA, and Census datasets. A head-to-head comparison with ZaliQL's classical estimators is reported in the evaluation section (Section 7.5).

The rest of the paper is organized as follows. In Section 2, we present the background and related work. Section 3 presents some definitions, and Section 4 introduces our causal inference and counterfactual operators. In Section 5, we present the system organization of PsiQL, followed by implementation details in Section 6. Experimental evaluation is detailed in Section 7. Then, we discuss some limitations and future work in Section 8 and conclusions in Section 9.

2. Background and Related Work

In this section, we present some necessary background and discuss related work.

2.1. Causal Inference Fundamentals

The fundamental problem of causal inference is well known: predictive models discover correlations but cannot answer interventional questions [19–21]. A model predicting loan defaults from income correlates features with outcomes but cannot say whether increasing income *causes* a higher approval rate—because the same confounders (e.g., wealth) may simultaneously cause both higher income and higher approval. Causal inference is

the process of establishing whether there is a cause-and-effect relationship between two variables, that is, determining if one variable causes a change in another variable. For example, if a person has a headache, she can either take a medication or not. If she takes the medication and then she gets better, there is no way to know whether she would have gotten better without the medication. In other words, we can only observe one outcome (i.e., of taking or not taking the medication), but not both. This is the fundamental problem of causal inference [22].

Potential outcomes (Rubin). Potential outcomes is a framework [23,24] for causal inference that formally specifies what a causal effect means for individuals and for different treatment and control groups. In this model, each individual or unit i with treatment T_i has two potential outcomes, $Y_i(1)$ and $Y_i(0)$, corresponding to treatment and control. Only one is observed: $Y_i = T_i Y_i(1) + (1 - T_i) Y_i(0)$.

Since individual causal effects cannot be measured directly, the average treatment effect (ATE) is used. ATE is calculated as $\tau = \mathbb{E}[Y(1) - Y(0)]$ and the conditional average treatment effect (CATE) is $\tau(x) = \mathbb{E}[Y(1) - Y(0) \mid X = x]$. Pearl's structural causal model (SCM) framework [19] provides an equivalent graphical formulation; both arrive at the same identification conditions.

Identification assumptions. Unconfoundedness ($\{Y(0), Y(1)\} \perp T \mid X$) suggests that given the adjustment set X, treatment assignment is as good as random. Positivity ($0 < P(T = 1|X = x) < 1$ for all x) entails that both treated and control units exist at every covariate value. SUTVA necessitates that there is no interference and no hidden treatment versions. These assumptions cannot be verified from data alone, rather they require domain knowledge.

Classical estimators. Propensity score matching [25] weights or matches units on $\hat{p}(x) = P(T = 1|X = x)$ to balance treated and control groups. IPW re-weights outcomes by $1/\hat{p}$ and $1/(1 - \hat{p})$. Stratification divides the sample into propensity-homogeneous strata. Stuart [26] reviews these methods; see also [19,22–24,27] for the broader identification and estimation literature. These classical estimators return a single global ATE and are sensitive to propensity model misspecification [27].

Meta-learners. Meta-learners [17] reduce causal estimation to supervised learning of nuisance functions, enabling flexible non-parametric heterogeneous effect estimation. The *S-learner* trains a single model on $(X, T) \to Y$ and reads off the effect as $\hat{f}(x, 1) - \hat{f}(x, 0)$. The *T-learner* trains separate models for treated and control units and takes their difference. The *X-learner* [17] imputes missing potential outcomes, and then blends individual-level effects with propensity-score weights. The *R-learner* [28] uses the Robinson decomposition to regress residualized outcomes on residualized treatment, achieving near-oracle rates. The *DR-learner* [29] applies double robustness: it is consistent if either the outcome model or the propensity model is correctly specified, but not necessarily both. Causal forests [30] generalize random forests to honest CATE estimation.

Uplift modeling. Uplift models directly estimate the incremental effect of a treatment on a binary outcome—the probability gain from intervening. For binary outcomes, uplift $= P(Y = 1|T = 1, X = x) - P(Y = 1|T = 0, X = x)$ is precisely the CATE. Uplift models are common in marketing (identifying customers who respond only when targeted) and clinical trials (identifying patients who benefit from treatment). The system exposes uplift models via the `USING_MODEL` clause.

Cross-fitting and diagnostics. Cross-fitting [31] trains nuisance models (propensity, outcome) on held-out folds, reducing regularization bias in the final effect estimate—especially important with flexible learners like XGBoost. Standardized mean difference (SMD) checks covariate balance: $\text{SMD}_j = |\bar{X}_j^{(1)} - \bar{X}_j^{(0)}|/s_{\text{pooled}}$; a threshold of 0.10 is conventionally used [26]. An SMD > 0.10 conventionally flags poor balance and warrants caution: it is

a diagnostic signal rather than an automatic veto of every design-supported claim (e.g., TWINS at max SMD = 0.119 remains a valid causal/quasi-experimental tier under the twin-pair design; Section 7.3). The system surfaces SMD via the DIAGNOSTICS parameter.

Covariate roles and conditioning. Given a target causal effect $T \rightarrow Y$, any other variable in the analyst's schema plays one of four roles relative to that edge: *confounder*, *mediator*, *collider*, or *instrument*. Each role dictates a different conditioning rule. Figure 1 shows all four roles in a single DAG, using the HMDA loan-approval scenario as a concrete anchor. The same analyst-chosen variables appear in the `ADJUSTING` clause of the SQL surface (Section 4.4), so the figure is also the template the PsiQL binder validates against: any variable placed in `ADJUSTING` is treated as a confounder, while mediators, colliders, and instruments must be handled via separate clauses or left unconditioned.

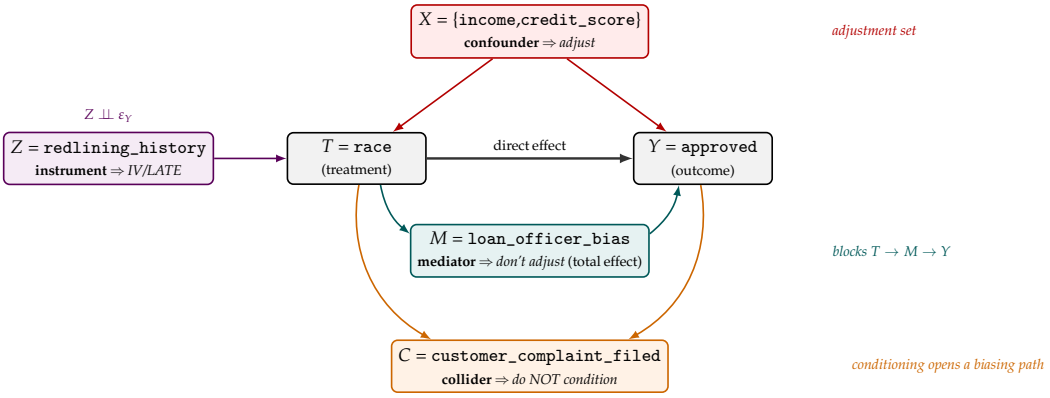


Figure 1. Covariate roles around the target causal edge $T \rightarrow Y$, shown in a single DAG using the HMDA loan-approval scenario. The treatment T is `race` and the outcome Y is `approved`; every other variable in the analyst's schema plays one of four roles relative to that edge. Confounder X (red; here `income` and `credit_score`) points into both T and Y and must be placed in the `ADJUSTING` clause. Mediator M (teal; here `loan_officer_bias`) sits on the causal path $T \rightarrow M \rightarrow Y$; if the analyst targets the total effect of T on Y, M must be left unconditioned (conditioning would block the very path we are trying to measure). Collider C (orange; here `customer_complaint_filed`) has both T and Y as parents; conditioning on C opens a spurious back-door between T and Y and inflates bias. Instrument Z (violet; here historical `redlining_history`) affects T but is independent of the error in Y; it identifies the local average treatment effect under monotonicity [32]. A single variable can play different roles in different queries, so the PsiQL binder validates role declarations against the schema rather than assuming them.

2.2. Counterfactual Reasoning and Algorithmic Recourse

Counterfactual explanations. Given a model $\hat{f}$ and an input x with $\hat{f}(x) = 0$ (e.g., loan denied), a counterfactual x' is the closest input that gets a different prediction ($\hat{f}(x') = 1$) and satisfies domain constraints. SHAP and LIME tell us which features mattered; counterfactuals tell us what to change and by how much [19,20,33–37]. Mittelstadt et al. [34] argue that this is the form of explanation most aligned with GDPR Article 22's right to meaningful information about automated decisions.

Desiderata. A good counterfactual should be proximate (few and small feature changes); feasible (satisfies domain constraints: immutability for age/race, permitted ranges for income, monotonicity for education); diverse (multiple alternatives, not all the same); and causally grounded (changes should correspond to actions the user can actually take). The tension between proximity and diversity is a key design challenge [35].

DiCE. Diverse Counterfactual Explanations [35] uses gradient-based or genetic optimization to produce a set of k diverse counterfactuals that all flip the prediction. DiCE balances proximity, diversity, and feasibility via a composite loss, and supports feature-level constraints (immutability, permitted ranges).

Feasibility and actionability. Immutability constraints (age, race cannot change), permitted ranges (income in $[30k, 200k]$), and monotonicity constraints (education can only increase) filter out counterfactuals that, while mathematically valid, are practically unachievable. Joshi et al. [36] and Ustun et al. [37] demonstrate that feasibility-constrained recourse is necessary for counterfactuals to be useful in practice. Prosperi et al. [38] extend this argument to clinical settings.

2.3. Proximity Search Index

The KD-tree [39] is a binary space-partitioning tree based index structure for nearest-neighbor search in Euclidean space, with $O(\log N)$ query time. It is well-suited for continuous covariate spaces with moderate dimensionality ($d < 20$). For high-dimensional or categorical feature spaces, MinHash LSH (Locality Sensitive Hashing) [40,41] provides approximate structural similarity, as it projects inputs into hash buckets such that similar items collide with high probability. LSH scales to billion-record datasets [42].

2.4. SQL-Based Causal Inference Systems

Table 1 summarizes the feature-level differences among the SQL-based frameworks for causal inference systems, including our approach, PsiQL.

Table 1. Feature comparison of SQL-based causal inference systems.

Capability	ZaliQL	CaRL	PsiQL
Causal Estimation			
Classical estimators (PSM/IPW/Strat.)	✓	✓	✓
Meta-learners (S/T/X/R/DR)	–	–	✓
Uplift models/Causal forests	–	–	✓
Auto-ensemble with stability index	–	–	✓
Cross-fitting (K-fold)	–	–	✓
Bootstrap confidence intervals	–	–	✓
Counterfactual Generation			
DiCE-based generation	–	–	✓
KD-Tree proximity search	–	–	✓
LSH structural similarity	–	–	✓
Constraint enforcement	–	–	✓
SQL Integration			
Declarative SQL-dialect operators	✓(SQL)	Datalog	✓
Multi-table JOIN resolution	–	✓	✓
WHERE/ORDER/LIMIT pushdowns	partial	–	✓
Table-valued relational output	partial	–	✓
GROUP BY slicing for CATEs	–	–	✓
Diagnostics			
SMD/covariate balance	limited	–	✓
Propensity overlap check	–	–	✓

Notes: ✓ = supported; – = not supported/not exposed; "partial" = limited support. Category labels (Causal Estimation, Counterfactual Generation, SQL Integration, Diagnostics) are section headers within the table, not italicized footnotes.

ZaliQL [11,12] pioneered SQL-based causal inference by expressing matching, IPW, and stratification as standard SQL queries inside PostgreSQL, demonstrating scalability to hundreds of millions of records. The limitation is expressiveness: the propensity model must be logistic regression (via MADlib), and the output is a single global ATE.

CaRL [13] uses Datalog-like rules for causal inference over multi-table relational data. Its strength is handling complex relational structures; its limitation is the non-SQL surface and no counterfactual generation.

Roy and Suciu [43] and Meliou et al. [44] formalize database query explanations via provenance—a fundamentally different sense of "why" than statistical causal inference. DoWhy [10] and CausalML [8] provide Python-based causal pipelines but operate outside the database.

2.5. SQL Extensions and Analytical Operators

SQL has expanded beyond simple querying through analytical extensions such as window functions, recursive queries, aggregations, and user-defined functions [45,46]. Modern engines such as DuckDB provide vectorized execution and extensible function registries for analytical workloads [47], while frameworks like DaskDB support distributed execution over Python dataframes [48].

Despite these advances, SQL remains descriptive. It can select, filter, aggregate, and transform—but it cannot represent interventions, enforce causal assumptions, or generate counterfactuals. Analysts must break out of SQL to perform causal modeling, destroying provenance and bypassing relational optimization.

2.6. Machine Learning Inside Databases

There is a substantial body of work on pushing ML into the database engine [14,49–51]. In-database ML systems provide native relational surfaces for training and scoring predictive models. MADlib [14] offers in-database regression, classification, clustering, and statistical functions on PostgreSQL and Greenplum; BigQuery ML [52] exposes logistic regression, boosted trees, deep neural nets, and matrix factorization through a SQL-native `CREATE MODEL` surface on Google's cloud data warehouse; SAP HANA PAL [53] bundles over 100 predictive and time-series algorithms in a single in-memory engine; and Microsoft SQL Server ML Services [54] embeds R and Python runtimes inside the database engine so that stored procedures can invoke ML pipelines with zero data movement. Karanasos et al. [50] extend SQL query processing with ML inference, and Makrynioti et al. [51] propose sql4ml as a declarative workflow. All four industrial systems excel at prediction—classification, regression, clustering, time-series forecasting—but none exposes average- or conditional-treatment-effect estimators, propensity diagnostics, overlap checks, or counterfactual-feasibility operators. PsiQL therefore complements rather than competes with this body of work: it adds a causal and counterfactual surface on top of a SQL dialect, and any of the four could, in principle, host a future port of ψ and φ as user-defined table-valued operators.

3. Problem Formulation and Definitions

3.1. Causal Effect Estimation (ATE/CATE, Uplift, and Meta-Learners)

We work over an observational dataset represented as a relation produced by a SQL query. The query's FROM/JOIN/WHERE clauses define the analysis population; projections define the outcome Y, treatment T, and covariates X; and optional `GROUP BY` clauses name slices S for subgroup reporting (Rule 4, Section 4).

Definition 1. *Potential outcomes. For unit i with covariates $\mathbf{X}_i$, binary treatment $T_i \in \{0,1\}$, and outcome Y_i, define potential outcomes $Y_i(1)$ and $Y_i(0)$. Only one is observed: $Y_i = T_i \cdot Y_i(1) + (1 - T_i) \cdot Y_i(0)$.*

Definition 2. *Average treatment effect.*

$$\tau = \mathbb{E}\big[Y(1) - Y(0)\big] \tag{1}$$

Definition 3. *Conditional average treatment effect/uplift.*

$$\tau(x) = \mathbb{E}\big[Y(1) - Y(0) \mid X = x\big] \tag{2}$$

In binary-outcome settings, "uplift" $= P(Y = 1|T = 1, X = x) - P(Y = 1|T = 0, X = x)$ *is the CATE for binary outcomes.*

Assumption 1. *Unconfoundedness/ignorability.*

$$\{Y(0), Y(1)\} \perp T \mid \mathbf{X}. \tag{3}$$

Assumption 2. *Positivity.*

$$0 < \Pr(T = 1 \mid \mathbf{X} = x) < 1 \quad \textit{for all } x \textit{ in the support of } \mathbf{X}. \tag{4}$$

Definition 4. *Propensity score.*

$$\hat{p}(\mathbf{X}_i) = \Pr(T_i = 1 \mid \mathbf{X}_i), \tag{5}$$

estimated via a flexible learner; our system defaults to XGBoost.

Definition 5. *Meta-learners. S-learner: single model on* $(X, T) \to Y$*; effect* $= \hat{f}(x, 1) - \hat{f}(x, 0)$*. T-learner: separate models for treated/control; effect* $= \hat{f}_1(x) - \hat{f}_0(x)$*. X-learner: imputes missing potential outcomes, then blends with propensity [17]. R-learner: residual-on-residual regression via Robinson decomposition [28,55], debiased via cross-fitting [31]. DR-learner: combines outcome modeling and IPW for double robustness [29].*

3.2. Counterfactual Reasoning (Feasibility and Actionability)

Definition 6. *Counterfactual explanation. Given instance* x *with predicted outcome* $\hat{y} = f(x)$*, a counterfactual* x' *satisfies (1)* $f(x') \neq \hat{y}$ *(or* $f(x') = y^*$*), (2)* x' *is close to* x *(proximity), (3)* x' *satisfies feasibility constraints C. Constraint types:* Immutability*:* $x'_j = x_j$ *for* $j \in I$ *(e.g., age, race);* Permitted Range*:* $x'_j \in [l_j, u_j]$*;* Monotonicity*:* $x'_j \geq x_j$ *(increase only) or* $x'_j \leq x_j$ *(decrease only).*

The counterfactual operator φ returns, for each input tuple, a set of feasible alternatives:

$$\begin{aligned} \varphi_{\{X_i,T,Y,M\}}(\mathcal{R}) = \big\{(t, \text{CFSet}(t)) \mid t \in \mathcal{R}, \\ \text{CFSet}(t) \subseteq \mathcal{N}(t) \cap \text{CausalValid}\big\} \end{aligned} \tag{6}$$

4. Causal Inference and Counterfactual Operators in PsiQL

4.1. Formal Operator Semantics

We introduce two operators, ψ and φ, for causal and counterfactual inference. In the next sections, we discuss their formal semantics, SQL extensions to support them and examples.

4.1.1. The ψ Operator (Causal Inference)

Let $\mathcal{R}$ be a relation with schema $\{ID, X_1, \ldots, X_d, T, Y\}$ produced by the FROM/JOIN/WHERE clauses. The causal inference operator returns a summary relation:

$$\psi_{\{X_i,T,Y,M\}}(\mathcal{R}) \mapsto \mathcal{R}_{\text{out}}, \tag{7}$$

where each row of $\mathcal{R}_{\text{out}}$ corresponds to a population stratum and carries columns (`ATE`, `CATE`, `ci_low`, `ci_high`, `SMD_max`, `model`).

4.1.2. The φ Operator (Counterfactual Generation)

$$\varphi_{\{X_i,T,Y,M\}}(\mathcal{R}) = \{(t, \text{CFSet}(t)) \mid t \in \mathcal{R}, \text{CFSet}(t) \subseteq \mathcal{N}(t) \cap \mathcal{V}\}, \tag{8}$$

where $\mathcal{N}(t)$ denotes the neighbor set from local search, $\mathcal{V}$ denotes the set of causally valid candidates (satisfying constraints), and CausalValid filters by constraints. Candidates are ranked by a composite score:

$$\text{Score}(c) = w_{\text{kd}} \cdot \text{NormDist}_{\text{KD}}(c) + w_{\text{lsh}} \cdot (1 - \text{NormSim}_{\text{LSH}}(c)). \tag{9}$$

4.2. Query Plan Integration

A causal query is translated into a *causal query plan* (CQP) that extends the standard relational plan with causal operators. For the ψ operator, the formal plan is

$$\text{CQP}_\psi = \tau_{\text{out}}(\psi_{\mathbf{X},T,Y,M,\Theta}(\sigma_{\text{pred}}(\bowtie_{\text{keys}}(\mathcal{R}_1, \mathcal{R}_2, \ldots)))) \tag{10}$$

where $\bowtie_{\text{keys}}$ is the multi-table join resolved by the MultiTableLoader, σ_{pred} applies WHERE predicates as a filter pushdown, $\psi_{\mathbf{X},T,Y,M,\Theta}$ is the causal estimation operator parameterized by adjustment set $\mathbf{X}$, treatment T, outcome Y, model M, and hyperparameters Θ, and τ_{out} is a final projection/LIMIT operator that materializes the result relation.

For the φ operator, the plan is analogous, with ψ replaced by $\varphi_{\mathbf{X},T,Y,M,C}$ where C is the constraint set (immutability, permitted ranges, monotonicity):

$$\text{CQP}_\varphi = \tau_{\text{out}}(\varphi_{\mathbf{X},T,Y,M,C}(\sigma_{\text{pred}}(\bowtie_{\text{keys}}(\mathcal{R}_1, \ldots)))) \tag{11}$$

Figure 2 shows the query plan tree for the HMDA running example.

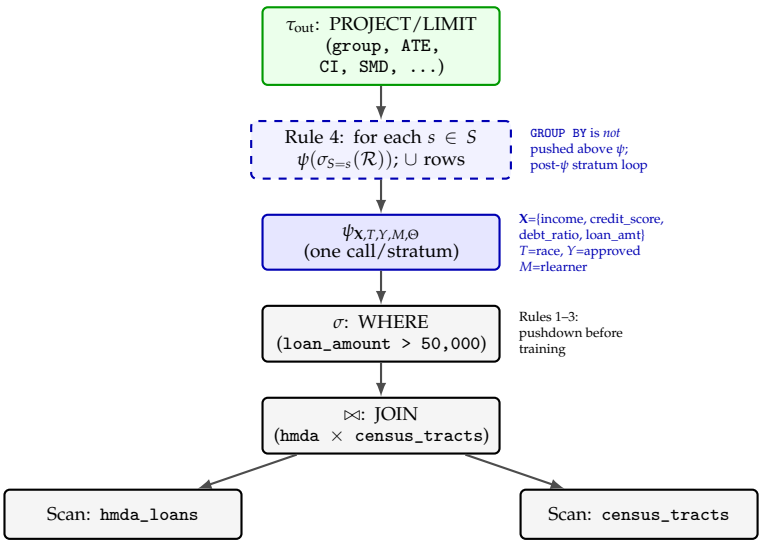


Figure 2. Causal query plan (CQP) for the HMDA running example with optional `GROUP BY`. Reading bottom-up: scans, join, selection pushdown (Rules 1–3), then ψ. When grouping is requested, Rule 4 wraps ψ in a post-ψ stratum loop (one ψ call per group s, then union) rather than a single-pass rewrite above ψ. Color convention: green = result, blue = causal/Rule 4, gray = standard RA.

WHERE predicates are pushed *below* the causal operator, reducing the analysis population before model training. `GROUP BY` is *not* pushed above ψ in a single pass; instead, Rule 4 applies a post-ψ stratum loop (one ψ invocation per stratum). ORDER BY and LIMIT apply after materialization and do not affect causal computation.

4.2.1. Rewrite Rules

We give three algebraic equivalences that let a cost-based optimizer move standard relational operators around the causal operator ψ without changing the identified estimand.

1. **Selection pushdown (admissible for T-independent predicates).** For a predicate φ that references only covariates $X \setminus \{T\}$,

$$\sigma_{\varphi}\big(\psi_{X,T,Y,M,\Theta}(R)\big) \;\equiv\; \psi_{X,T,Y,M,\Theta}\big(\sigma_{\varphi}(R)\big),$$

   provided the propensity score $e(x) = \Pr(T = 1 \mid X = x)$ remains strictly bounded away from 0 and 1 on the filtered subpopulation (i.e., overlap is preserved). Predicates that reference T are not commutable: filtering on the treatment before ψ changes which units define the treated and control arms, altering the identified estimand from an ATE over the full population to a conditional effect that the analyst may not have requested.
2. Projection pullup. If the projection π_A retains all columns of $X \cup \{T, Y\}$, then

$$\pi_A\big(\psi_{X,T,Y,M,\Theta}(R)\big) \;\equiv\; \psi_{X,T,Y,M,\Theta}\big(\pi_{A \cup X \cup \{T,Y\}}(R)\big).$$

   This equivalence lets the optimizer minimize intermediate column width without altering the causal identification: anything the user ultimately projects away can be dropped earlier, as long as the adjustment set, treatment, and outcome survive into the causal phase.
3. **Join commutativity with ψ (covariate enrichment).** For a join $R \bowtie_{\theta} S$ where S contributes additional covariates that are *pretreatment* (measured before T is assigned),

$$\psi_{X \cup X_S,\, T,\, Y}\big(R \bowtie_{\theta} S\big) \;\equiv\; \psi \text{ applied after the join.}$$

   Adjustment variables from S may be freely joined before or after the operator as long as they are pretreatment; post-treatment columns from S must be excluded from the covariate set to avoid post-treatment bias (conditioning on a descendant of T can partially block the very causal path the operator is trying to estimate).

These rules let a cost-based optimizer move $\sigma / \pi / \bowtie$ around ψ without changing the causal estimand, provided the overlap and pretreatment conditions hold. A full cost model and plan-space search are beyond the scope of this paper and deferred to future work.

4.2.2. Rule 4 (GROUP BY as a Post-ψ Stratum Loop)

Rules 1–3 license moving selection, projection, and joins relative to ψ. `GROUP BY` is different: it is not a single-pass relational rewrite above ψ, but a post-ψ stratum loop. Let S be a grouping attribute and $\mathcal{R}_s = \sigma_{S=s}(\mathcal{R})$. Then,

$$\gamma_S\big(\psi_{X,T,Y,M,\Theta}(\mathcal{R})\big) \;\equiv\; \bigcup_{s \in \pi_S(\mathcal{R})} \big\{\, (s,\, \psi_{X,T,Y,M,\Theta}(\mathcal{R}_s)) \,\big\},$$

provided each stratum $\mathcal{R}_s$ satisfies positivity ($0 < e(x) < 1$ on the support of $\mathcal{R}_s$) and has both treatment arms present. Operationally (cf. Algorithm 1), the relational phase materializes $\mathcal{R}$ via joins, filters, and projections only; `GROUP BY` then invokes ψ once per stratum and unions the per-stratum result rows.

Algorithm 1 `EXPLAIN_CAUSALLY_WHY` execution pipeline

Require: Extended SQL query Q, data source D
Ensure: Causal result relation $\mathcal{R}_{\text{out}}$
Parse Q via SQLGlot $\rightarrow$ extract $(Y, T, \mathbf{X}, M, \Theta,$ FROM/JOIN/WHERE; optional GROUP BY for Rule 4)
Resolve FROM/JOIN via MultiTableLoader $\rightarrow$ unified relation $\mathcal{R}$
Apply WHERE pushdown: $\mathcal{R} \leftarrow \sigma_{\text{pred}}(\mathcal{R})$
Validate schema: check $Y, T, \mathbf{X}$ exist; verify treatment variation
if $M =$ `auto_ensemble` **then**
  models $\leftarrow$ parse `MODELS` parameter
  for all model m_i in models **(parallel) do**
    Train m_i on $\mathcal{R}$; compute ATE_i
  end for
  $\text{ATE} \leftarrow \text{aggregate}(\text{ATE}_1, \ldots, \text{ATE}_k)$ via `ENSEMBLE_RULE`
  stability $\leftarrow 1 - \text{range}/(|\text{mean}| + \text{range} + \epsilon)$
else
  Train single model M on $\mathcal{R}$
  Compute ATE, per-row CATE
end if
if `CROSS_FIT` enabled **then**
  Split $\mathcal{R}$ into K folds; cross-fit nuisance models; aggregate
end if
if `RETURN_CI` = 1 **then**
  Bootstrap B iterations (evaluation default $B = 50$; see Section 7.2) $\rightarrow$ (ci_low, ci_high)
end if
if `DIAGNOSTICS` = 1 **then**
  Compute SMD for each covariate; compute propensity overlap
end if
if GROUP BY specified **then**
  FOR each stratum $s \in \pi_S(\mathcal{R})$: $\mathcal{R}_s \leftarrow \sigma_{S=s}(\mathcal{R})$; invoke ψ on $\mathcal{R}_s$; union rows
end if
Materialize results as DataFrame $\rightarrow$ register as SQL relation
return $\mathcal{R}_{\text{out}}$

4.2.3. φ Under `GROUP BY`

The current design does *not* support stratum-specific counterfactual sets: `GROUP BY` with φ does not emit a separate CFSet per group. Counterfactual generation runs on the full (WHERE-filtered) population; any grouping clause is ignored for CF materialization (the engine may still compute auxiliary per-group ATE diagnostics, but these do not partition the returned CF relation). Analysts who need group-restricted alternatives should push the stratum into `WHERE` (or a subquery) before φ.

4.2.4. Admissibility of Rewrite Rules

Rules 1–4 are admissible in the sense that, under the stated side conditions (overlap/positivity preservation for selection and strata; retention of $X \cup \{T, Y\}$ for projection; pretreatment covariates for joins), each equivalence preserves the identified estimand of ψ (ATE or stratum-wise CATE). They do not invent new identification results; they only license moving $\sigma / \pi / \bowtie / \gamma_S$ relative to ψ without changing what is being estimated.

Relational pushdown streamlines execution by shrinking the analysis population before training, but it does *not* relax unconfoundedness, positivity, consistency, or SUTVA—those identification assumptions remain the analyst's responsibility regardless of plan shape. The foregoing is an operational contract (typed treatments, schemas, rewrite Rules 1–4, and the validity/admissibility bridge): full denotational semantics for ψ and φ in the potential-outcomes + relational-algebra sense is future work.

4.2.5. Treatment Typing

Treatments are typed as (i) binary $T \in \{0,1\}$, (ii) multi-valued categorical $T \in \{t_1, \ldots, t_K\}$, or (iii) continuous $T \in \mathbb{R}$. The meta-learners consume a binary encoding $\iota(T) \mapsto \{0,1\}$: for binary T, ι is the identity; for multi-valued categorical T, the analyst (or a derived column) selects a contrast of interest (e.g., one level vs. the rest); for continuous T, ι discretizes at a threshold (default: median) with a minimum-support check. In the HMDA experiments, multi-level race is evaluated via the binary indicator `is_white` rather than a multi-arm encoding. Categorical covariates in $\mathbf{X}$ are one-hot encoded before estimation.

4.3. Running Example—HMDA Fair Lending

We use this example throughout the paper to illustrate how data flows through the operator pipeline. Table 2 shows the six input records. The two questions we answer: (Q1) does race affect loan approval after controlling for income, credit score, debt ratio, and loan amount? (Q2) what minimal feasible change would flip applicant 2's denial to approval?

Table 2. Step 1—Input relation `hmda_loans` (6 records).

ID	Income	Credit	DTI	Race	Loan	Appr.
1	85,000	720	0.28	White	250,000	1
2	62,000	680	0.35	Black	180,000	**0**
3	95,000	750	0.22	White	300,000	1
4	58,000	640	0.42	Hispanic	150,000	0
5	78,000	710	0.30	Black	220,000	1
6	70,000	690	0.38	White	200,000	0

Columns: Credit = `credit_score`; DTI = `debt_ratio`; Loan = `loan_amt`; Appr. = `approved`.

Step 2—Causal query (ψ) as a relational-algebra plan. Question Q1 (does `race` affect `approved` on larger loans, after adjusting for the financial covariates?) is expressed as the ψ plan in Figure 3. Reading the tree bottom-up: a base Scan materializes `hmda_loans`; a projection π drops every column not needed for the causal identification (keeping only the adjustment set, the treatment `race`, and the outcome `approved`); a selection σ restricts to loans above $50,000 (the analysis population); and the ψ operator then applies the R-learner to estimate ATE with covariates X = {income, credit_score, debt_ratio, loan_amt}, treatment T = `race`, and outcome Y=`approved`. The final τ_{out} projects only the columns the user wants in the result (ATE, CI, SMD, model). This plan is deliberately written in RA so that readers can follow the computation before the SQL surface is introduced in Section 4.4; the textual PsiQL SQL that compiles to this exact tree is shown in that section.

Step 3—ψ output relation (Table 3): Each row is a population slice; columns are ATE, confidence interval, and max SMD.

Step 4—Counterfactual query (φ) as a relational-algebra plan. Question Q2 (what minimal feasible change flips applicant 2's denial?) is expressed as the φ plan in Figure 4. The structure mirrors the ψ plan of Figure 3, with three differences that every counterfactual query exhibits: (i) the selection σ narrows the input to the audited subgroup—here, the denied Black applicants—because counterfactuals are per-instance whereas ψ is a population-level estimator; (ii) the causal node is replaced by $\varphi_{X,T,Y,\Pi,K}$, where Π is the immutability/permitted-range constraint set and K is the number of alternatives requested; (iii) τ_{out} emits a set of feasible alternatives per row instead of a single effect, so its schema carries CF vectors, feasibility flags, and proximity (L_1) metrics. The plan is again written in RA first; the textual PsiQL SQL that compiles to it is given in Section 4.4.

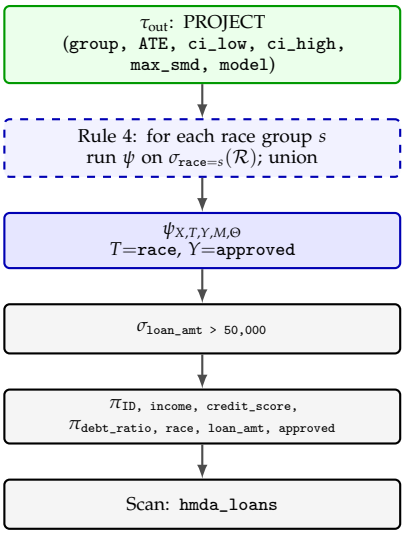


Figure 3. Q1 as a relational-algebra tree for ψ when `GROUP BY race` is requested. Rule 4 inserts a post-ψ stratum loop (dashed); the relational phase below ψ is unchanged. τ_{out} columns match the typed ψ output schema (Section 4.4). Multi-level `race` is illustrative; reported HMDA experiments use binary `is_white`.

Table 3. Step 3—ψ output when `GROUP BY race` is requested (illustrative per-stratum CATE rows under Rule 4; not the global binary-`is_white` ATE reported in Section 7.3).

Group	ATE	ci_Low	ci_High	max_smd	Model
White	+0.012	−0.008	0.032	0.12	rlearner
Black	−0.015	−0.042	0.012	0.18	rlearner
Hispanic	−0.008	−0.028	0.012	0.15	rlearner

Illustrative; actual values from HMDA evaluation in Section 7.

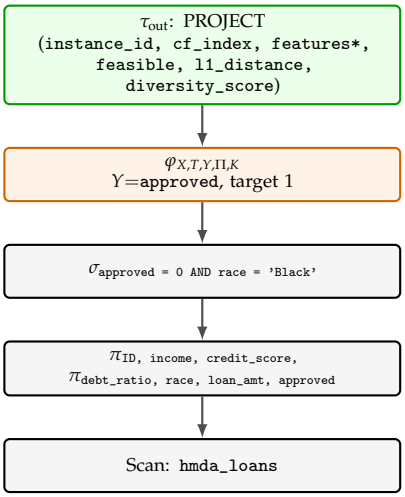


Figure 4. Q2 as a relational-algebra tree for φ. τ_{out} schema matches the typed φ output schema in Section 4.4 (flattened one row per counterfactual). In the output node, the asterisk in `features*` denotes the full counterfactual feature vector (equivalently features* in the typed schema).

Step 5—φ output relation (Table 4): Each row is a feasible alternative for applicant 2; columns show the changed features, feasibility, L1 distance, and predicted outcome.

Table 4. Step 5—φ output for applicant 2 (illustrative). Column layout is wide-table style; engine output uses `instance_id`, `cf_index`, feature columns, `feasible`, `l1_distance`, `diversity_score` per Table 5.

Inst._id	cf	Income	Credit	DTI	Feas.	L_1	Div.
2	1	72,000	680	0.30	✓	0.22	0.41
2	2	68,500	695	0.28	✓	0.18	0.38
2	3	75,000	670	0.32	✓	0.31	0.45

Illustrative; race is immutable; income capped at $120 k per constraint.

Step 6—From RA to SQL. We have now laid out the full pipeline — input table, ψ plan, ψ output, φ plan, φ output — without assuming any SQL-level familiarity with PsiQL. The very next section (Section 4.4) introduces the textual SQL surface and shows the exact PsiQL SQL that compiles to Figures 3 and 4; readers may prefer to read Section 4.4 first and return here for the semantic grounding.

4.4. SQL Operator Surface

Running Example, Q1 and Q2 in PsiQL SQL Dialect

The relational-algebra plans in Figures 3 and 4 map directly onto the following SQL statements, making the RA ↔ SQL bridge explicit:

```
{-- Q1:} psi for ATE of race on approval among larger loans
SELECT *
FROM EXPLAIN_CAUSALLY_WHY(
      (SELECT ID, income, credit_score, debt_ratio, race,
            loan_amt, approved
       FROM hmda_loans
       WHERE loan_amt > 50000)
      TREATMENT race
      OUTCOME  approved
      METHOD   R_LEARNER
      LIMIT_CI TRUE
);

-- Q2: phi for counterfactuals on denied Black applicants
SELECT *
FROM EXPLAIN_COUNTERFACTUAL(
      (SELECT ID, income, credit_score, debt_ratio, race,
            loan_amt, approved
       FROM hmda_loans
       WHERE approved = 0 AND race = 'Black')
      OUTCOME  approved
      TARGET   1
      TOTAL_CFS 3
      IMMUTABLE race
);
```

These two statements are the textual surface corresponding to the ψ and φ nodes in the plans; the engine lowers them to DuckDB [47] using SQLGlot [56], then dispatches to the causal and counterfactual engines described in Section 5.

Our PsiQL system extends the SQL dialect with two declarative operators—ψ (`EXPLAIN_CAUSALLY_WHY`) and φ (`EXPLAIN_COUNTERFACTUAL`)—as custom tokens. Formally, both operators are typed relational transformers: they accept a relation $\mathcal{R}$ produced by standard SQL (FROM/JOIN/WHERE) and return a new relation $\mathcal{R}_{\text{out}}$ whose schema includes causal quantities (ATE, CATE, CI, SMD, or counterfactual candidates). Optional `GROUP BY` for ψ follows Rule 4; for φ, stratum-specific CF sets are unsupported. This design makes causal results naturally composable with downstream SQL operations.

The ψ operator maps

$$\psi : \mathcal{R} \times (\mathbf{X}, T, Y, M, \Theta) \rightarrow \mathcal{R}_{\psi}$$

The φ operator maps

$$\varphi : \mathcal{R} \times (\mathbf{X}, T, Y, M, C) \rightarrow \mathcal{R}_{\varphi}$$

where $\mathbf{X}$ is the adjustment set, T is the treatment, Y is the outcome, M is the estimator, Θ is the hyperparameter set, and C is the constraint set (immutability, ranges, monotonicity).

The keywords `EXPLAIN_CAUSALLY_WHY`, `ADJUSTING`, `INTERVENE_WITH`, `USING_MODEL`, and `WITH_PARAMETERS` are custom tokens parsed by SQLGlot; no database engine accepts this syntax without the middleware. Adoption as first-class SQL would require standardization or engine-native registration as table-valued functions.

4.5. Syntax Overview

Extended generic SQL templates and the Advanced SQL Composition subsection appear in Appendix A; the material below is retained for the HMDA walkthrough and output-schema reference.

Both operators follow the same high-level structure:

```
SELECT EXPLAIN_CAUSALLY_WHY outcome_expr,
       ADJUSTING (covariate_1, ..., covariate_k),
       INTERVENE_WITH treatment_expr,
       [USING_MODEL model_name],
       [WITH_PARAMETERS (param_key = value, ...)]
FROM  ...
[JOIN ...]
[WHERE ...]
[GROUP BY ...];

SELECT EXPLAIN_COUNTERFACTUAL outcome_expr,
       ADJUSTING (covariate_1, ..., covariate_k),
       INTERVENE_WITH treatment_expr,
       [USING_MODEL model_name],
       [WITH_PARAMETERS (param_key = value, ...)]
FROM  ...
[JOIN ...]
[WHERE ...]
[GROUP BY ...];
```

φ-specific syntax (preferred template). The listing above mirrors ψ for binder uniformity; the counterfactual operator is more naturally written with outcome/target/count/immutability clauses, as in the running example:

```
SELECT * FROM EXPLAIN_COUNTERFACTUAL(
  (SELECT ... FROM ... [JOIN ...] [WHERE ...])
  OUTCOME   outcome_col
  TARGET    target_value
  TOTAL_CFS k
  IMMUTABLE col_1 [, col_2, ...]
);
```

Here, `OUTCOME` is Y, `TARGET` is the desired predicted label, `TOTAL_CFS` is the number of alternatives K, and `IMMUTABLE` lists features that must not change. `GROUP BY` is accepted syntactically but does not produce stratum-specific CF sets (see above).

Clause semantics.

`ADJUSTING`$(X_1, \ldots, X_k)$: Specifies the adjustment set $\mathbf{X} = \{X_1, \ldots, X_k\}$. Under the backdoor criterion [19], $\mathbf{X}$ blocks all backdoor paths from T to Y in the assumed causal DAG, satisfying unconfoundedness: $\{Y(0), Y(1)\} \perp T \mid \mathbf{X}$. The analyst is responsible for supplying a defensible adjustment set; the system validates only that the named columns exist in the source relation.

`INTERVENE_WITH` T: Identifies the treatment variable T. Formally, this defines the *intervention* $do(T = t)$ in Pearl's do-calculus [19]: the ATE is $\mathbb{E}[Y \mid do(T = 1)] - \mathbb{E}[Y \mid do(T = 0)]$. For binary $T \in \{0, 1\}$, the system compares $T = 1$ (treated) against $T = 0$ (control). The system validates that both values exist in the analysis population (positivity check).

`USING_MODEL` M: Selects the causal estimator. Supported models: `slearner`, `tlearner`, `xlearner`, `rlearner`, `drlearner`, `auto_ensemble`, among others. Each is a meta-learner that reduces causal estimation to supervised learning of nuisance functions using XGBoost as the base model.

`WITH_PARAMETERS`$(\ldots)$: Key–value pairs that configure the execution: `RETURN_CI=1` enables bootstrap confidence intervals, `DIAGNOSTICS=1` enables SMD/overlap diagnostics, `CROSS_FIT_FOLDS=`K enables K-fold cross-fitting, `CF_IMMUTABLE` specifies immutable features for counterfactual generation.

`FROM/JOIN/WHERE`: Define the analysis population $\mathcal{R} = \sigma_{\text{pred}}(\bowtie_{\text{keys}} (\mathcal{R}_1, \ldots))$ as a standard relational expression. The causal operators consume $\mathcal{R}$ as input; standard SQL optimizations (predicate pushdown, LIMIT) apply before causal execution.

`GROUP BY` S: Partitions $\mathcal{R}$ into strata $\mathcal{R}_s$ for each distinct value $s \in S$; the causal operator computes a separate ATE/CATE for each stratum. Formally: ψ returns $\{ (s, \tau_s) \mid s \in \pi_S(\mathcal{R}) \}$ where $\tau_s = \mathbb{E}[Y(1) - Y(0) \mid S{=}s]$. See Rule 4. For φ, stratum-specific CF sets are unsupported; restrict via `WHERE` instead.

Output types. `EXPLAIN_CAUSALLY_WHY` (ψ) returns a relation of type

$$\mathcal{R}_\psi : \langle \text{group}, \tau_{\text{ATE}}, \tau_{\text{CATE}}, \text{ci_low}, \text{ci_high}, \text{smd_max}, \text{model} \rangle$$

`EXPLAIN_COUNTERFACTUAL` (φ) returns a relation of type

$$\mathcal{R}_\varphi : \langle \text{instance_id}, \text{cf_index}, \text{features}^*, \text{feasible}, \text{l1_distance}, \text{diversity_score} \rangle$$

where features* denotes the full feature vector of the counterfactual candidate. Both are standard SQL relations and can be used in subsequent `JOIN`, `WHERE`, `GROUP BY`, and `ORDER BY` clauses.

Formal φ output schema (Table 5). Column names are canonical for engine materialization; wide feature columns are omitted from the table for space.

Table 5. φ (`EXPLAIN_COUNTERFACTUAL`) output columns.

Column	Type	Description
`instance_id`	key	Source row identifier
`cf_index`	int	Rank index of the CF within the instance ($1 \ldots K$)
`features*`	mixed	Placeholder for the wide per-feature columns materialized by the engine (the full CF feature vector features*)
`feasible`	bool	Constraint satisfaction flag
`l1_distance`	float	L_1 proximity to the original instance
`diversity_score`	float	Diversity relative to other CFs for the instance

4.6. EXPLAIN_CAUSALLY_WHY

EXPLAIN_CAUSALLY_WHY estimates treatment effects for the population defined by the relational query.

Given

- An outcome expression Y;
- A treatment variable T;
- A set of covariates X;
- A population defined through SQL (FROM, JOIN, WHERE); optional `GROUP BY` for ψ follows Rule 4;

the operator fits one or more causal models and returns

- Average treatment effect (ATE);
- Slice-level conditional treatment effects (CATEs);

- Optional bootstrap confidence intervals (when RETURN_CI = 1);
- Optional diagnostics (balance, overlap, learner stability) when DIAGNOSTICS = 1;
- Metadata about model choices and parameters.

ψ (`EXPLAIN_CAUSALLY_WHY`) returns columns `ATE`, `CATE`, and optionally `ci_low`/`ci_high` and `max_smd` (when `RETURN_CI = 1` and `DIAGNOSTICS=1` are enabled, respectively), plus `model` metadata. φ (`EXPLAIN_COUNTERFACTUAL`) returns `instance_id`, `cf_index`, the full feature vector of each counterfactual, and quality metrics: `feasible`, `l1_distance`, and `diversity_score`. Both output types are standard SQL relations.

The following query estimates treatment effects on mortality, drawing covariates from three joined tables with cross-fitting and bootstrap CIs.

Illustrative Causal (ψ) Query

```
SELECT EXPLAIN_CAUSALLY_WHY mortality_outcome,
      ADJUSTING (age, education_years, income_level,
                health_status, family_size, region),
      INTERVENE_WITH treatment_assignment,
      USING_MODEL meta_learners
      WITH_PARAMETERS (
        RETURN_CI=1,
        DIAGNOSTICS=1,
        SAMPLE_SIZE=5000,
        CROSS_FIT_FOLDS=5,
        BOOTSTRAP_SAMPLES=1000
      )
FROM patient_data p
JOIN demographic_info d ON p.patient_id = d.patient_id
JOIN treatment_history t ON p.patient_id = t.patient_id
WHERE p.age BETWEEN 18 AND 65
  AND d.region IN ('urban', 'suburban')
  AND t.treatment_date >= '2020-01-01'
ORDER BY p.age DESC
LIMIT 1000;
```

4.7. EXPLAIN_COUNTERFACTUAL

EXPLAIN_COUNTERFACTUAL generates actionable "what-if" alternatives for each instance in the result set. It is semantically distinct from EXPLAIN_CAUSALLY_WHY—this operator returns instance-level counterfactuals, not population-level treatment effects.

Workflow:

1. Fit model(s) to capture the relationships between features, treatment, and outcome.
2. Search for feasible alternative configurations that change the outcome.
3. Enforce constraints (immutability, permitted ranges, monotonicity).
4. Return a ranked set of counterfactual suggestions.

The resulting table includes:

- Proposed feature changes;
- Constraint satisfaction indicators;
- Proximity and diversity metrics.

The following query generates counterfactual loan outcomes with immutability, range, and monotonicity constraints.

Illustrative Counterfactual (φ) Query

```
SELECT EXPLAIN_COUNTERFACTUAL loan_default,
      ADJUSTING (credit_score, income, debt_ratio,
                employment_years, loan_amount),
      INTERVENE_WITH loan_approval,
      USING_MODEL meta_learners
      WITH_PARAMETERS (
        DICE_TOTAL_CFS=5,
```

```
        DICE_IMMUTABLE_FEATURES='credit_score',
    DICE_PERMITTED_RANGES=
      'income:50000-200000,debt_ratio:0.1-0.5',
        DICE_MONOTONE_FEATURES='income:increasing',
        SAMPLE_SIZE=1000,
        DIVERSITY_WEIGHT=0.5,
        PROXIMITY_WEIGHT=0.5
       )
FROM loan_applications l
JOIN borrower_profiles b
  ON l.borrower_id = b.borrower_id
JOIN credit_history c
  ON l.borrower_id = c.borrower_id
WHERE l.loan_amount > 100000
  AND b.employment_years >= 2
  AND c.credit_score >= 600
-- GROUP BY omitted: per-instance CFs (use WHERE to restrict cohort)
ORDER BY l.loan_amount DESC
LIMIT 500;
```

Note: `GROUP BY` is omitted in φ examples; stratum-specific CF sets are unsupported (restrict via `WHERE` instead).

4.8. Advanced SQL Composition

Representative CTE/subquery patterns are summarized in Appendix A. Because both operators are relational, they compose naturally with standard SQL constructs. CTEs and subqueries can be used in the FROM clause to prepare data, while the operators themselves must appear in the top-level SELECT. The system supports:

- CTEs in FROM clauses for data preparation;
- Subqueries in FROM clauses;
- Window functions in CTEs and subqueries;
- CASE expressions;
- GROUP BY/HAVING;
- ORDER BY with aggregates.

The following fragment constructs age-group-matched cohorts via CTEs, then estimates conditional treatment effects on 30-day mortality across hospital and insurance slices.

Illustrative Fragment

```
-- CTE defines age-matched cohorts; operator runs on the result
WITH cohorts AS (...), matched AS (...)
SELECT EXPLAIN_CAUSALLY_WHY mortality_30day,
      ADJUSTING (age, comorbidity_score, hospital_type, insurance_status),
      INTERVENE_WITH treatment_group,
      USING_MODEL auto_ensemble
      WITH_PARAMETERS (RETURN_CATE=1, RETURN_CI=1, CROSS_FIT_FOLDS=10)
FROM matched m JOIN patient_outcomes po ON ...
GROUP BY m.hospital_type, m.insurance_status;
```

The complete query appears in the project repository.

5. System Organization of PsiQL

The architecture of PsiQL involves three key components: a parser, a binder, and execution engines (causal engine and counterfactual engine). The parser recognizes causal operators in the SQL and extracts relational structure. The binder maps the resulting query plan to a causal specification. The engine trains models and returns results as standard SQL relations. Figure 5 shows the workflows involving the components of PsiQL. These components are described in detail in the following sections.

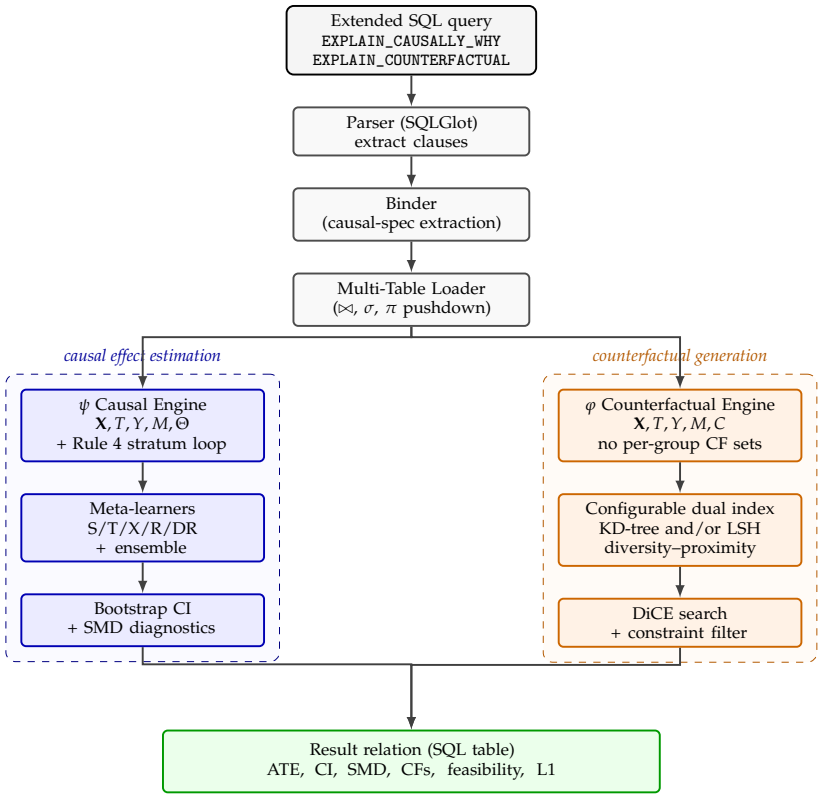


Figure 5. System architecture of PsiQL. The φ branch builds a configurable dual index (KD-tree and/or LSH); the shipped setting enables both as a diversity–proximity trade-off (not an unqualified quality default; see Section 7.4). Result relations expose typed ψ/φ columns (ATE/CI/SMD or `instance_id/l1_distance/diversity_score`).

5.1. Parser

A SQLGlot [56]-based parser is used to parse a SQL query with causal clauses. PsiQL leverages some components of DuckDB [47] for input/output and results materialization.

Causal operators are introduced as declarative SQL-dialect extensions. When a query containing `EXPLAIN_CAUSALLY_WHY` or `EXPLAIN_COUNTERFACTUAL` is submitted, the system parses the extended syntax and builds an abstract syntax tree. Standard clauses (`WHERE`, `JOIN`, projections) define the relational phase that materializes $\mathcal{R}$; optional `GROUP BY` for ψ follows Rule 4, while operator-specific clauses (`ADJUSTING`, `INTERVENE_WITH`, `USING_MODEL`, `WITH_PARAMETERS`) define the causal specification.

Causal inference is expressed declaratively and remains compatible with normal SQL optimization.

5.2. Binder

The binder converts a parsed query into a causal execution plan by aligning relational semantics with causal structure. Joins are resolved prior to modeling to construct the unit of analysis; filters determine the eligible population; projections identify available covariates, treatment variables, and outcomes. Optional `GROUP BY` is scheduled per Rule 4 (unsupported for per-group φ CF sets).

The binder extracts both standard SQL parameters and causal-specific parameters via the SQLGlot AST traversal. It then performs column existence validation before dispatching to the execution engine. Diagnostics such as positivity and overlap checks are performed during execution when the DIAGNOSTICS parameter is enabled. The binding is performed in Python 3.10 by traversing the SQLGlot AST, not by modifying DuckDB's internal logical plan.

5.3. Causal Engine

Once the binder identifies outcome, treatment, and covariates and the loader produces a unified relation, the causal engine takes over (the ψ branch in Figure 5). The causal engine executes treatment-effect estimation over the relational data Algorithm 1 details the `EXPLAIN_CAUSALLY_WHY` execution pipeline: a relational phase (joins/filters/projections) materializes $\mathcal{R}$, then a causal phase trains meta-learners and computes effects; optional `GROUP BY` follows Rule 4.

For each query, the engine returns:

- Average treatment effects (ATE);
- Slice-level conditional effects (CATE);
- Optional bootstrap confidence intervals (when RETURN_CI = 1);
- Optional diagnostics such as standardized mean differences and propensity overlap (when DIAGNOSTICS = 1).

All outputs are materialized as DataFrames that can be used as SQL relations, enabling downstream filtering, joining, visualization, and aggregation inside the database.

Connecting the steps of Algorithm 1. Line-by-line walkthroughs of Algorithms 1 and 2 appear in Appendix B.

The causal engine validates treatment variation (ensuring both treated and control groups exist) and implements meta-learners including S-, T-, X-, R-, and DR-learners [8,57]. Categorical features are one-hot encoded automatically. Continuous treatments can be discretized with minimum-support guard-rails, though we have only tested binary treatments in this paper. The R-learner and DR-learner implementations use cross-fitting to reduce overfitting in high-dimensional settings [31]. Data are split into K folds; nuisance models are trained on $K-1$ folds and evaluated on the held-out fold, and final estimates are obtained by aggregating cross-fitted predictions. The number of folds is controlled via the `CROSS_FIT_FOLDS` parameter.

The `auto_ensemble` mode trains multiple base learners (S-, T-, X-, R-, DR-learners) in parallel using CPU threads. The final ATEs (and, when requested, CATEs) are aggregated from individual model estimates using mean, median, or trimmed mean rules. A stability index summarizes dispersion across learners and exposes ensemble robustness to the user.

Diagnostics are off by default but easy to turn on. `RETURN_CI=1` runs non-parametric bootstrap and returns confidence intervals. `DIAGNOSTICS=1` computes SMD for covariate balance and propensity overlap for common support. These are the same checks an analyst would run in a Python 3.10 notebook but the difference is that here, they show up in the output table alongside the ATE, which makes them harder to ignore.

5.4. Counterfactual Engine

Counterfactual reasoning is implemented as an optimization-driven search enriched with dual indexing. The φ branch of Figure 5 illustrates the data flow from indexing to final ranking. Algorithm 2 details the counterfactual generation algorithm. Constraint filtering (Phase 4) enforces immutability, permitted ranges, and monotonicity as a post-optimization gate, ensuring that no infeasible candidate reaches the output. Candidates that violate constraints are filtered before scoring.

The counterfactual engine generates "what-if" alternatives for individual instances. Each feasible candidate is scored along three axes: proximity to the original instance, constraint-aware feasibility, and diversity relative to previously selected counterfactuals. The engine then ranks candidates by a composite score and returns the top-k results. The final output is a relational table that includes proposed feature changes, feasibility indicators, proximity metrics, and diversity scores for each instance. The final set of counterfactuals is returned as a relational table containing candidate features, feasibility

metrics, and diversity scores, making them naturally interoperable with downstream SQL operations.

Algorithm 2 Counterfactual generation algorithm (φ engine)

Require: Instance x, target outcome y^*, constraints C, count k, training data D, model $\hat{f}$
Ensure: Counterfactuals $\mathcal{CF}$
Phase 1 — Index construction
Build KD-tree index $\mathcal{T}$ on D for Euclidean proximity search
Build MinHashLSH index $\mathcal{L}$ on D for structural similarity search
Train causal model $\hat{f}$ on D (if not pre-trained)
Phase 2 — Candidate pool construction
$\mathcal{N}_{\text{KD}} \leftarrow$ query $\mathcal{T}$ with x: k-nearest neighbors by L2 distance
$\mathcal{N}_{\text{LSH}} \leftarrow$ query $\mathcal{L}$ with x: structurally similar instances
$\mathcal{P} \leftarrow \mathcal{N}_{\text{KD}} \cup \mathcal{N}_{\text{LSH}}$ (deduplicated)
Phase 3 — DiCE optimization
Run DiCE on $\hat{f}$, seeded with candidate pool $\mathcal{P}$, target y^*, constraints C
$\mathcal{CF}_{\text{raw}} \leftarrow$ DiCE output (up to DICE_TOTAL_CFS candidates)
Phase 4 — Constraint filtering and ranking
$\mathcal{CF} \leftarrow \emptyset$
for all candidate $c \in \mathcal{CF}_{\text{raw}}$ **do**
  if c satisfies all constraints C (immutable features, ranges, monotonicity) **then**
    $\text{score}(c) \leftarrow w_{\text{kd}} \cdot \text{NormDist}_{\text{KD}}(c) + w_{\text{lsh}} \cdot (1 - \text{NormSim}_{\text{LSH}}(c))$
    $\mathcal{CF} \leftarrow \mathcal{CF} \cup \{(c, \text{score}(c))\}$
  end if
end for
Sort $\mathcal{CF}$ by score; return top-k by joint proximity + diversity
return $\mathcal{CF}$

Dual indexing for counterfactual search. Generating realistic counterfactuals is a constrained search problem: find the closest point in feature space that (a) changes the predicted outcome and (b) satisfies all domain constraints. The naive approach of searching the full training set exhaustively is $O(N \cdot d)$ per instance and does not scale. DiCE [35] uses gradient-based optimization to escape this, but when seeded randomly, it finds points that are close to the decision boundary in model space but far from the query instance in feature space, producing unrealistic recourse (e.g., suggesting a 40-year-old applicant becomes a 22-year-old to get a loan approved).

We propose a dual-index design to address this problem structurally. It maintains two complementary views of "closeness":

1. KD-tree (geometric proximity): Finds instances that are close in Euclidean space—same financial profile, similar demographics. These are the most proximate alternatives in the continuous feature subspace. KD-tree query time is $O(\log N + k)$, making it practical at inference time.
2. MinHash LSH (structural similarity): Finds instances that share the same pattern of feature values, even if they differ geometrically. For example, two applicants with very different incomes but the same credit score, loan type, and debt bracket are structurally similar and are good recourse candidates. LSH operates on categorical/hashed representations in $O(m)$ time where m is the number of hash functions.

Neither index is sufficient alone. KD-tree misses structurally similar instances along categorical or sparse dimensions—Euclidean distance treats a change from "White" to "Black" the same as a change from income \$60 k to \$61 k. LSH returns noisy candidates without geometric grounding. The union $\mathcal{N}_{\text{KD}} \cup \mathcal{N}_{\text{LSH}}$ gives DiCE a candidate pool that is both geometrically proximate and structurally diverse. The empirical trade-off this design produces (more CFs with lower L_1 for KD-tree alone vs. fewer CFs with higher

structural sparsity and diversity for KD-tree+LSH) is quantified in the dual-index ablation in Section 7.4; we defer the numbers to that section so that the design rationale here is read in one continuous block.

Connecting the steps of Algorithm 2.

5.5. Complexity Analysis

Table 6 summarizes the time and space complexity of key phases. The SQL parsing overhead is negligible (<0.1 s in all experiments). JOIN resolution is dominated by the pandas merge operation. Model training accounts for 85–99% of total wall-clock time across all datasets evaluated.

Table 6. Complexity analysis of key execution phases.

Phase	Time	Space
SQL parsing (SQLGlot)	$O(\|Q\|)$	$O(\|Q\|)$
JOIN resolution	$O(N \cdot M)$	$O(N + M)$
Meta-learner training	$O(N \cdot d \cdot n_{\text{est}})$	$O(N \cdot d)$
Cross-fitting (K folds)	$K \times O(N \cdot d \cdot n_{\text{est}}/K)$	$O(N \cdot d)$
Bootstrap CI (B iterations)	$B \times O(N \cdot d)$	$O(B)$
KD-Tree build	$O(N \log N)$	$O(N)$
KD-Tree query	$O(\log N + k)$	$O(k)$
LSH build (m permutations)	$O(N \cdot m)$	$O(N \cdot m)$
LSH query	$O(m)$	$O(k)$
Constraint filtering	$O(\|\text{CF}\| \cdot \|C\|)$	$O(\|\text{CF}\|)$

5.6. Additional Features

5.6.1. Multi-Table Support

Causal tasks often depend on data distributed across multiple tables. (e.g., outcomes in one table, covariates and treatments in others). The multi-table loader of PsiQL resolves these joins before causal execution. The parser parses the query and exposes the `FROM` and `JOIN` clauses and then the loader looks up the referenced tables in a registry of in-memory DataFrames or file-backed sources.

Equi-joins are executed using pandas merge operations [58] on the identified join keys. Non-equi joins are implemented as post-merge filters: the loader performs a broader merge and then applies the non-equi predicates as boolean masks. WHERE, ORDER BY, and LIMIT pushdowns are applied to the unified DataFrame before it is passed to the causal or counterfactual engine.

5.6.2. Extensibility

The parser, binder, and inference engine are all replaceable. Adding a new meta-learner means implementing one Python 3.10 class with a `fit/predict` interface; adding a new SQL operator means registering a token with the parser.

6. Implementation

SQLGlot [56] is used to parse SQL in pure Python 3.10, supports custom token registration without modifying the core grammar, and integrates naturally with our CausalML-based [8] execution engine. The rest of the stack is straightforward. XGBoost [57] serves as the default base learner, scikit-learn [59] handles preprocessing and one-hot encoding, pandas [58] and NumPy [60] provide vectorized computation, and DuckDB materializes the final results as SQL relations. The prototype and all experiments reported in this paper were implemented in Python 3.10.

Hyperparameters and Reproducibility

Full hyperparameter defaults and reproducibility notes are consolidated in Appendix B.

7. Evaluation

In our experimental evaluation, we consider three questions:

- Q1 (Expressiveness): Can SQL-dialect operators express realistic causal and counterfactual analyses that normally require multi-language pipelines?
- Q2 (Statistical quality): Do the resulting estimates align with established causal baselines on standard datasets?
- Q3 (Systems behavior): How do runtime and resource usage scale across datasets and model configurations?

We focus on high-stakes domains where causal questions are central: fair lending, algorithmic fairness, and healthcare, plus a large-scale Census case study.

7.1. Datasets

Table 7 summarizes the real-world impact datasets used in our experiments. Together, they cover regulatory, social, and clinical settings, and exercise different parts of the system: multi-table joins, high-cardinality covariates, and moderately large samples.

Table 7. Datasets.

Dataset	Records	Feats.	Domain	Treatment T	Outcome Y
HMDA	8,976,748	100+	Financial	`derived_race`	`action_taken`
COMPAS [61]	7215	50+	Justice	`is_white`	`two_year_recid`
TWINS [62]	11,984	30+	Healthcare	`T` (birth weight)	`yf` (mortality)
Census [63] *	378,817	11+	Economic	`is_white`	`employed`

* Real ACS PUMS California (2018, 1-Year person) has 378,817 records. In Section 7.10, the synthetic ACS-like row ($N \approx 9423$, ATE ≈ 0.115, SMD ≈ 0.031) is a balanced synthetic benchmark; the companion real-data run ($N = 10{,}000$, 50-iteration bootstrap; ATE $= 0.005$, 95% CI $[-0.013, 0.013]$, max SMD $= 0.377$, runtime ≈ 234 s) is a real-data diagnostic that flags unconfoundedness/positivity failure rather than a policy-grade employment effect.

Our HMDA subsample ($N = 5000$) is drawn uniformly at random from the 2019 California HMDA extract after restricting to conventional first-lien home-purchase loans, to keep the per-experiment wall-clock budget within the single-laptop constraint. Scaling beyond $N = 100{,}000$ is supported by the engine (see Section 7.7, median runtime 18.86 s at $N = 100{,}000$), and the full 8.97M-record HMDA panel is compatible with the pipeline; we defer that run to future work because the point of the HMDA experiment here is to exercise the diagnostic path (SMD flagging, imbalance narrative) rather than to provide a policy-grade effect estimate.

7.2. Experimental Setup and Baselines

The experiments are run in Python 3.10 on a single M2 MacBook Air with 16 GB RAM.

Estimators. We do not introduce new causal estimators, but rather, the engine wraps the same CausalML-style meta-learners (S/T/X/R/DR) used in typical Python 3.10 notebooks. The empirical focus is whether those estimators behave sensibly when driven from SQL (Q2) and how execution time scales with N and model configuration (Q3)—not micro-benchmarks of CSV export glue code, which are negligible relative to training once bootstrap CIs or ensembles are enabled.

Configuration parameters. All experiments use `numpy.random.seed(42)` and `sklearn` train/test splits with `test_size=0.2` (80/20 split, stratified by treatment). Default XG-

Boost hyperparameters across all datasets: `n_estimators=100`, `max_depth=6`, `learning_rate=0.1`. Throughout the evaluation (Table 8 and Algorithm 1), the default bootstrap budget is $B = 50$—a compute-bound choice on single-node hardware, well below the conventional $B \geq 1000$ for tight percentile/BCa intervals, trading CI precision for feasibility. Counterfactual evaluation instances are sampled stratified by treatment (50% treated, 50% control) from the test split to ensure both groups are represented. Data leakage is avoided by building KD-tree and LSH indices on the training split only; query instances are drawn from the held-out test split.

For TWINS, COMPAS, and synthetic Census, we use $B = 50$ bootstrap iterations per dataset. This is below the convention of $B \geq 1000$ for tight percentile or BCa intervals; at $B = 50$, the Monte Carlo standard deviation of the 2.5th/97.5th empirical quantiles is on the order of 0.02, which is non-negligible relative to the point estimates. We chose $B = 50$ after measuring that $B = 1000$ on TWINS ($N = 11{,}984$, fold count 5, per-replicate ≈ 65 s) would take ≈ 18 h, exceeding the practical compute/wall-clock budget on single-node hardware. Readers should interpret the reported CIs as indicative and the direction/ordering of effects as the primary evidence; a follow-up study with $B \geq 500$ is planned.

Table 8 summarizes, for each experiment, the dataset (real or synthetic), sample size, estimator, bootstrap budget, whether a CI is reported, and the evaluation subsection where results are discussed.

Table 8. Experimental setting protocol matrix for all Evaluation runs. "CI?" indicates whether a bootstrap CI is reported.

Experiment	Dataset	*N*	Estimator	*B*	CI?	Notes
Main benchmark	COMPAS (real)	3000	auto-ensemble	50	Y	Section 7.3
Main benchmark	TWINS (real)	11,984	auto-ensemble	50	Y	Section 7.3
Main benchmark	Census (synth)	9423	auto-ensemble	50	Y	Balanced synthetic benchmark
Main benchmark	Census (real)	10,000	R-learner	50	Y	Real-data diagnostic
Main benchmark	HMDA (real)	5000	auto-ensemble	10 †	N	SMD diagnostic path
ZaliQL comparison	four datasets	3 k–5 k	IPW/Strat./PSM/ens.	–	N	Section 7.5
CF quality	COMPAS (real)	100	R-learner + DiCE	–	N	Section 7.8
CF quality	TWINS (real)	100	R-learner + DiCE	–	N	Section 7.8 ($N = 100$)
Dual-index ablation	COMPAS (real)	100	R-learner + DiCE	1000 ‡	Y	Section 7.4
Dual-index probe	COMPAS (real)	20	R-learner + DiCE	–	N	Section 7.4 ($N = 20$)
CATE subgroups	COMPAS / TWINS	full/3 k	R-learner	varies	Y/N	Section 7.6

† HMDA: 10-iteration min/max range, not a 95% CI. ‡ Bootstrap $B = 1000$ percentile CIs on mean L_1/sparsity/diversity for the $N = 100$ dual-index arms.

7.3. Causal Accuracy and Diagnostics

Identification assumptions. For each dataset, we assume a DAG where $T \to Y$ with observed confounders $\mathbf{X}$, unconfoundedness, positivity, and consistency; run protocols are in Table 8, and claim strength is mapped in Table 9. On TWINS, the twin-pair design provides strong support for unconfoundedness [62]. On COMPAS, unconfoundedness is approximated by conditioning on available arrest and demographic covariates; unmeasured factors remain a threat (exploratory disparity tier). On Census and HMDA, we treat effects as *diagnostic* associations when SMD diagnostics flag imbalance.

Interpretation tiers for causal outputs. Because identification strength varies across datasets, we classify each reported ATE into one of four claim tiers (Table 9) rather than treating every numeric output as equally actionable.

Table 10 spans these settings: TWINS (valid causal claim), COMPAS (exploratory disparity), HMDA and real ACS (fairness diagnostic only), and synthetic Census (balanced synthetic benchmark).

Table 10 presents the results. All runs use the same meta-learner configurations as CausalML, with diagnostics enabled.

Table 9. Claim tiers for interpreting causal SQL outputs. Tiers separate policy-grade causal claims from exploratory disparity audits and diagnostics-only runs.

Tier	When	Required Diagnostics	Paper Example
Valid causal claim	Quasi-experimental design; CI excludes 0; balance adequate *or* only marginally above SMD 0.10 when the design supports unconfoundedness	CI + stability	TWINS (SMD 0.119)
Exploratory disparity	Observational fairness audit; CI may cross 0; weak balance	SMD + learner breakdown	COMPAS
Fairness diagnostic only	Imbalance/overlap failure; ATE not for policy	SMD flag; mute/omit CI	HMDA, Census real ACS
Synthetic benchmark	Synthetic data with adequate balance; not a policy claim	Balance check	Census (synth.)

Table 10. Benchmark results from actual executions (SQL-dialect pipeline). TWINS, COMPAS, and Census (synthetic) use auto-ensemble with 50 bootstrap samples for CI; HMDA CI omitted due to covariate imbalance (see SMD diagnostics). The synthetic ACS row (N = 9423, ATE $\approx$ 0.115, SMD $\approx$ 0.031) is a balanced synthetic benchmark. The real ACS PUMS row (^{P}, N = 10,000, ATE $\approx$ 0.005, SMD $\approx$ 0.377; R-learner, not auto-ensemble) is a real-data diagnostic: CI crosses zero and max SMD $\gg$ 0.10 shows unconfoundedness/positivity failure on this specification.

Dataset	ATE	95% CI	Runtime (s)	N	Max SMD
COMPAS	−0.045	(−0.070, 0.023) *	237.4	3000	0.335 §
TWINS	−0.103	(−0.123, −0.088)	3232.4	11,984	0.119
US Census (real ACS) ^{P}	0.005	(−0.013, 0.013) **	233.7	10,000	0.377 §§
US Census (synthetic) ‡‡	0.115	(0.085, 0.117)	475.4	9423	0.031
HMDA	−0.010	–	29.2	5000	0.190 †

* COMPAS 95% CI crosses zero; the effect is not statistically significant at the 95% level. § COMPAS max SMD $>$ 0.10 indicates substantial covariate imbalance; ATE should be interpreted with caution. ‡‡ Synthetic ACS-like benchmark (9423 records, auto-ensemble); see Section 7.10. † HMDA SMD $>$ 0.10 indicates moderate imbalance; ATE should be interpreted with caution. At N = 5000, a full bootstrap pass takes $\approx$ 230 s; scaling to the conventional $B \geq 500$ would require $\approx$ 32 h on the same single-node hardware. Where a bootstrap spread is cited for HMDA, we therefore report the 10-iteration min/max range rather than labeling it a 95% CI. Future work will revisit HMDA with a larger bootstrap or an influence-function-based analytic CI. ^{P} Real ACS PUMS California (N = 10,000, 50-iteration bootstrap; *R-learner*, whereas the synthetic row uses auto-ensemble). ** CI crosses zero; the real-data effect is not statistically significant. §§ Max SMD 0.377 $\gg$ 0.10 ($\sim$12$\times$ the synthetic SMD of 0.031): severe covariate imbalance; the $\sim$23$\times$ gap between synthetic ATE 0.115 and real ATE 0.005 is an ATE ratio, not an SMD ratio.

Results on the TWINS dataset are straightforward to interpret. The system recovers a protective treatment effect on mortality (ATE ≈ -0.10) with marginal covariate balance (max SMD $= 0.119$, just above the 0.10 threshold). Our TWINS ATE of -0.103 is within the range reported in ML-causal follow-up work [17] that preprocesses the twins sample aggressively (e.g., filtering to low-birth-weight pairs or to specific covariate subsets). The raw mortality contrast in Almond, Chay & Lee [62] is smaller, on the order of -0.02 to -0.03, so this value should be read as sample-specific rather than a canonical consensus; we emphasize the relative ordering across learners and the sign of the effect rather than the magnitude.

The COMPAS results require careful interpretation. Here, the treatment is `is_white` $\in \{0, 1\}$, and the outcome is `two_year_recid`. The 95% bootstrap CI (-0.070 to 0.023) crosses zero, so the effect is not statistically significant at $\alpha = 0.05$; max SMD = 0.335 exceeds the 0.10 balance threshold (covariate balance is questionable); and the ensemble stability index is low ($\approx$0.099), reflecting learner disagreement under stressed overlap. The point ATE of -0.045 ($\sim$4–5 pp) is therefore indicative only (exploratory disparity tier): directionally consistent with ProPublica's disparity audit [61,64], but not a confirmed racial disparity or causal effect. We use a binary `is_white` indicator rather than multi-level race encoding, following ProPublica's design [61] and avoiding the identification complexities of multi-valued treatments. Race is obviously not reducible to a binary [65], but the potential-outcomes framework requires it.

Census (synthetic) has good balance (max SMD = 0.031) and ATE = 0.115 under auto-ensemble; real ACS (R-learner) is the diagnostic counterpart under severe imbalance (max SMD = 0.377). HMDA results show moderate imbalance (max SMD = 0.190, well above the 0.10 threshold). With `DIAGNOSTICS=1`, the system flags this rather than silently returning an ATE; an analyst would need propensity trimming to $[0.1, 0.9]$ before interpreting the results.

7.4. Ablation Study with the Dual-Index

Table 11 reports a smaller $N = 20$ comparison of KD-tree alone vs. the combined KD-tree+LSH strategy on COMPAS. The dual-index configuration is a configurable trade-off via per-index toggles (`USE_KDETREE`, `USE_LSH`): the shipped default enables both indices and favors manifold/structural diversity at a substantial proximity cost; KD-tree-only search can return near-identical neighbors (many instances with $L_1 = 0$).

Table 11. Dual-index ablation on 20 COMPAS instances (R-learner, 3 CFs requested). Combined KD-tree + LSH raises mean L_1 by $\sim$18$\times$ (0.47$\rightarrow$8.67) and sparsity by $\sim$4$\times$ (0.37$\rightarrow$1.50), with runtime +14% (13.6$\rightarrow$15.5 s). Pairwise diversity is *not* measured in this $N = 20$ table. L_1 denotes the mean $\sum_j |\Delta x_j|$ over changed features on the raw encoded scale (unnormalized). The main comparison is Table 12.

Strategy	Avg CFs	Avg L1	Avg Sparsity	Avg RT (s)
KD-tree only	8.05	0.47	0.37	13.6
KD-tree + LSH	6.80	8.67	1.50	15.5

Table 12 is the primary dual-index comparison: $N = 100$ COMPAS instances, arms B (KD-tree+DiCE) vs. D (KD+LSH combined), with bootstrap $B = 1000$ percentile 95% CIs on mean L_1, sparsity, and pairwise diversity. L_1 is raw unnormalized $\sum_j |\Delta x_j|$ on encoded features. KD+LSH raises mean L_1 by $\sim$21$\times$ (0.41$\rightarrow$8.80) and diversity by $\sim$15$\times$ (0.42$\rightarrow$6.50) relative to KD-only; the CIs do not overlap. The $N = 20$ table remains as an earlier probe (means only, no diversity/CIs).

Table 12. Dual-index ablation on $N = 100$ COMPAS instances (R-learner; arms B vs. D; bootstrap $B = 1000$). L_1 is raw unnormalized.

Arm	*N*	Avg CFs	Feas	L_1 (95% CI)	Sparsity (95% CI)	Diversity (95% CI)	RT
B_KDTree_DiCE	100	8.44	1.00	0.41 [0.30, 0.55]	0.35 [0.26, 0.45]	0.42 [0.30, 0.58]	6.1 s
D_Combined (KD+LSH)	100	6.25	0.96	8.80 [7.85, 9.76]	1.57 [1.47, 1.67]	6.50 [5.76, 7.27]	7.9 s

7.5. Comparison with Existing Systems

We reproduced the published ZaliQL stack (PostgreSQL 9.6 + MADlib 1.10) and the three classical estimators that ZaliQL exposes through SQL—inverse-probability weighting (IPW, Hájek-normalized with propensity clipping), stratification (NTILE-5 over MADlib

propensity scores), and propensity-score matching (`matchit_propensity_score`, $k = 1$, caliper 0.2)—on the same $N = 3000$–5000 splits used elsewhere in this paper. Table 13 reports the result; Table 1 summarizes the feature-level differences.

Functionally, ZaliQL exposes IPW/Stratification/PSM and a single global ATE, whereas PsiQL additionally exposes meta-learner ensembles, bootstrap CIs, per-row CATEs, SMD/overlap diagnostics, and a counterfactual operator (none of which are available inside ZaliQL's Postgres-9.6 stack). On the four datasets, the two systems agree on direction for IPW and Stratification; magnitudes match closely on TWINS/HMDA (within$\sim$0.001) and more loosely on Census (within$\sim$0.02), as expected when both implement textbook estimators with different normalization/stratification details. In Table 13, the classical IPW/Strat/PSM "PsiQL RT" entries are Python 3.10/scikit-learn reference baselines (not in-engine), while auto-ensemble RT is PsiQL's in-engine wall-clock. On COMPAS/TWINS/HMDA, those Python 3.10 classical references are often faster than ZaliQL Docker, but on Census, the same classical rows are 11–18$\times$ *slower* than ZaliQL Docker (e.g., IPW 55.67 s vs. 3.15 s). Full-sample TWINS/Census end-to-end times remain far from interactive latency (Section 7.7).

Table 13. Head-to-head comparison on identical N and splits. Column semantics: ZaliQL RT = wall-clock seconds for a single end-to-end run of the published ZaliQL/MADlib Docker stack (PostgreSQL 9.6 + MADlib 1.10); **PsiQL RT (classical IPW/Strat/PSM)** = Python 3.10/scikit-learn reference baseline wall-clock on the same splits (not an in-engine SQL path); **PsiQL RT (auto-ensemble)** = in-engine PsiQL auto-ensemble wall-clock (S/T/X/R/DR learners). All runs use a single CPU laptop without GPU.

Dataset	Method	ZaliQL/MADlib ATE	ZaliQL RT (s)	PsiQL ATE	PsiQL RT (s)
COMPAS	IPW	−0.0035	9.29	+0.0060	1.48
	Stratification	−0.0246	2.98	−0.0236	2.05
	PSM	−0.2466	8.54	−0.0197	2.82
	auto-ensemble	–	–	−0.0453	0.21
TWINS	IPW	−0.1205	3.14	−0.1206	0.99
	Stratification	−0.1210	4.47	−0.1210	1.19
	PSM	−0.0970	4.16	−0.0858	1.26
	auto-ensemble	–	–	−0.1341	0.33
HMDA	IPW	−0.0352	3.04	−0.0352	1.77
	Stratification	−0.0341	3.46	−0.0340	2.24
	PSM	−0.0406	6.21	+0.0080	2.29
	auto-ensemble	–	–	−0.0099	5.24
Census	IPW	+0.0997	3.15	+0.0791	55.67
	Stratification	+0.0995	3.54	+0.0833	58.63
	PSM	+0.1004	3.96	+0.0712	46.25
	auto-ensemble	–	–	+0.0748	18.74

ZaliQL ATE values are computed by the published MADlib operators; classical PsiQL ATE/RT rows are Python 3.10/scikit-learn reference baselines on the same N, while auto-ensemble ATE/RT is in-engine. Across IPW and Stratification, the two systems agree to within $\sim$0.001 on TWINS and HMDA, within $\sim$0.02 on Census (e.g., IPW +0.0997 vs. +0.0791), and within $\sim$0.022 on COMPAS—the residual difference being traceable to ZaliQL's Hájek normalization and NTILE-5 stratification choices. The PSM gap on COMPAS and HMDA is expected: ZaliQL's 1:1 matching with caliper 0.2 keeps only the matched subsample (e.g., 872/3000 on COMPAS, 1997/5000 on HMDA), whereas the Python 3.10 reference PSM uses the full sample. The auto-ensemble row has no ZaliQL counterpart because ZaliQL does not expose a meta-learner ensemble. Census classical reference RTs are 11–18$\times$ slower than ZaliQL Docker and are not in-engine PsiQL timings.

The TWINS benchmark [17,62] reports protective ATEs of −0.08 to −0.12. Our bootstrap-aggregated auto-ensemble gives −0.103, within that range. On COMPAS, the −0.045 ATE is directionally consistent with ProPublica's disparity audit [61,64], but the CI crosses zero and the estimate is not statistically significant.

Ensemble stability. The auto-ensemble model breakdown on COMPAS shows: T-learner −0.048, S-learner −0.044, X-learner −0.047, and DR-learner +0.082. The auto-ensemble gives −0.045, effectively averaging out the DR-learner outlier. On TWINS, the

full-sample auto-ensemble (−0.103, Table 10) is less extreme than single-pass learners on the 5 k head-to-head subsample (≈−0.134), illustrating where ensembling and bootstrap aggregation help.

The DR-learner's sign reversal on COMPAS (+0.082 versus −0.044 to −0.048 for the other three learners) is consistent with near-violation of the overlap assumption (max SMD = 0.335, stability index = 0.099/≈0.10): IPW-heavy estimators become unstable when propensity scores push toward 0 or 1. Following standard practice under low overlap [66], we treat the reported auto-ensemble ATE as indicative rather than inferential and recommend that practitioners either (i) trim the sample to the overlap region, (ii) exclude the DR-learner when the stability index falls below a pre-registered threshold (we use 0.3 as a rule of thumb), or (iii) report the per-learner breakdown and let the analyst choose. The PsiQL auto-ensemble surface exposes the per-learner breakdown and flags low stability via a diagnostic return column.

Diagnostics behave correctly: TWINS max SMD (0.119) marginally exceeds the 0.10 balance threshold, which the system flags; Census (synthetic) has excellent balance (SMD = 0.031); and HMDA is flagged for moderate imbalance (SMD = 0.190). This is exactly what a handwritten Python 3.10 pipeline would do, minus the data-export ceremony.

7.6. Conditional Treatment Effects by Subgroup

Table 14 reports CATEs for the COMPAS recidivism analysis (treatment: `is_white`; outcome: `two_year_recid`) broken down by age group and sex, using the R-learner.

Table 14. CATE by subgroup on COMPAS (R-learner). Subgroup CIs from a bootstrap CATE sweep; global ATE from the subgroup evaluation on a matched sample.

Slice	Group	ATE	CI_{low}	CI_{high}	*N*
Age group	Young (<30)	−0.036	−0.092	−0.013	3373
	Middle (30–49)	0.023	−0.011	0.091	2863
	Senior (≥50)	−0.054	−0.079	0.105	978
Sex	Male	0.017	−0.039	0.018	5819
	Female	0.001	−0.041	0.098	1395
Global (all) §		−0.033	–	–	3000

§ Global ATE (−0.033) is the R-learner on a 3000-record COMPAS sample; Table 10 (−0.045) uses the auto-ensemble with bootstrap CI. Subgroup *N*s sum to 7214 (full dataset); the global row uses the 3000-record sample for consistency with the main benchmark. The Male subgroup CI-high (0.018) being within 0.001 of the point estimate (0.017) reflects a tight bootstrap distribution with skew toward the lower tail; it is not a typo.

Each subgroup analysis uses a WHERE clause filtering to the subgroup of interest. The middle-aged subgroup (30–49) shows a *positive* point CATE (+0.023), reversing sign relative to the young and senior groups (−0.036, −0.054). However, the 95% bootstrap CI (−0.011 to 0.091) crosses zero, so this is a non-finding rather than a robust sign reversal. The sex-stratified CATEs are near zero (+0.017 for males, 0.001 for females), with CIs that also cross zero. The broader point is that a global ATE of −0.033 (R-learner) masks directionally conflicting heterogeneity, which is exactly what subgroup slicing is designed to surface.

Table 15 shows similar results for TWINS by birth order.

Table 15. CATE by birth order on TWINS (R-learner).

Subgroup	ATE	*N*	Max SMD
First-born (bord = 1)	−0.131	5587	0.119
Second-born (bord = 0)	−0.088	6397	0.099
Global	−0.102	11,984	0.109

Global R-learner ATE (−0.102) is consistent with the auto-ensemble ATE in Table 10 (−0.103). Both fall within the published benchmark range (−0.08 to −0.12). First-born max SMD (0.119) marginally exceeds the 0.10 balance threshold; global max SMD (0.109) is also slightly above, indicating that subgroup-level covariate balance should be monitored.

7.7. Performance and Scalability

We measure end-to-end runtime, including parsing, loading, model training, and result materialization. Figure 6 visualizes runtime and resource usage across datasets and configurations; Table 16 provides the underlying numbers.

Table 16. Runtime and performance by dataset (SQL-dialect pipeline).

Dataset	Runtime (s)	Records	Sample	ATE	CI Width	Max SMD
COMPAS	237.4	7215	3000	−0.045	0.092	0.335
TWINS	3232.4	11,984	11,984	−0.103	0.035	0.119
US Census (50-iter cap) ‡‡	475.4	9423	9423	0.115	0.032	0.031
HMDA†	29.2	5000	5000	−0.010	–	0.190 †

‡‡ Census: synthetic ACS-like benchmark. Real ACS PUMS diagnostic (N = 10,000, ATE 0.005, max SMD 0.377) omitted here for space; see Table 10. † HMDA: moderate covariate imbalance; ATE should be interpreted with caution.

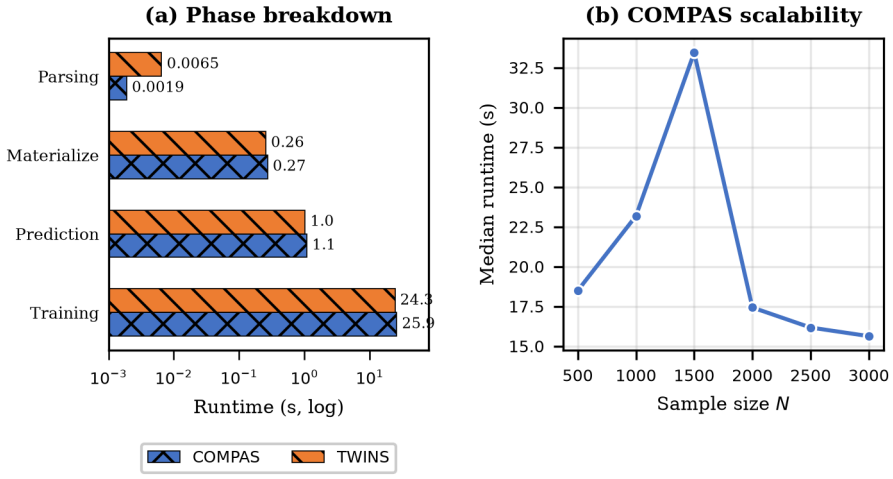


Figure 6. Performance analysis of the PsiQL ψ pipeline from the evaluation pipeline. Panel (**a**)—phase-level runtime (log scale): end-to-end runtime decomposed into the four pipeline phases (SQL parsing via SQLGlot, nuisance-model training, prediction, and result materialization) on two representative datasets: COMPAS at N = 3000 and TWINS at N = 5000 (the TWINS panel is a profiling subsample; full-sample TWINS timing is in Table 16). Training dominates at >95% of wall-clock time; parsing is below 0.01 s and not visible on a linear axis, which motivates the log scale so that all four phases remain legible at once. Panel (**b**)—COMPAS scalability: median runtime vs. sample size N over three trials per N. The curve flattens above $N \approx$ 1000–1500 because per-run setup costs (parsing, cross-fit splits, loader materialization) amortize over larger batches, so the pipeline exhibits roughly constant per-row cost beyond that regime. Together, the two panels show that PsiQL's overhead on top of the underlying meta-learner is bounded by parsing and materialization, not causal estimation—extending SQL with ψ adds no measurable cost beyond the model training the analyst would have run anyway. The panel (a) legend is placed below the axes so it does not overlap the bars.

Figure 6a shows the phase-level breakdown on a log scale for COMPAS (N = 3000) and TWINS (N = 5000—the profiling script subsamples TWINS for this panel): training

dominates at 24–26 s (>95% of total time), prediction adds ≈ 1.1 s, materialization ≈ 0.3 s, and SQL parsing is <0.01 s. The log axis makes all phases visible simultaneously—a feature obscured in linear-scale plots where training dwarfs everything else. Figure 6b shows that median COMPAS runtime is largely flat once N exceeds ∼1000–1500 under this configuration, consistent with fixed per-run setup costs amortizing over larger batches.

Runtime is driven more by model complexity than by N. For COMPAS single meta-learners; it ranges ≈ 0.2–11 s per method (auto-ensemble fastest; R-learner slowest), while Census (9423 samples, many covariates, ensemble + diagnostics) takes ≈475 s after capping bootstrap at 50 iterations. TWINS at full $N = 11{,}984$ takes ≈3232 s. On the larger HMDA dataset, median runtime scales from 1.02 s at $N = 1000$ to 18.86 s at N = 100,000 (extended scalability runs), demonstrating sub-linear scaling. Training and cross-fitting account for 85–99% of total time; SQLGlot parsing and DuckDB integration are negligible (<0.1 s). All experiments used a single CPU laptop without a GPU; TWINS and Census end-to-end wall times (≈3232 s and ≈475 s, respectively) are far from interactive latency—consistent with reading Table 13 as a functional comparison rather than an interactive-speed claim (Section 7.5).

7.8. Counterfactual Evaluation

We evaluate EXPLAIN_COUNTERFACTUAL on the COMPAS, TWINS, and HMDA datasets to assess the quality and performance of the counterfactual generation pipeline (see Table 17). The counterfactual engine combines DiCE optimization with dual-index search (KD-tree for proximity, MinHash LSH for structural similarity) and constraint filtering, as described in Section 5.4.

Table 17. Counterfactual quality metrics (DICE_TOTAL_CFS = 3 requested for COMPAS/TWINS, 10 for HMDA; R-learner; CF_IMMUTABLE = age for COMPAS, `dfageq` for TWINS). COMPAS and TWINS use N = 100 held-out instances with per-CF delta metrics. HMDA uses N = 15 instances. TWINS uses PCA (22d→10d, 98% variance retained). The `CFs/inst` column reports total candidates surviving the constraint filter, which includes the KD-tree/LSH seed pool returned alongside DiCE output; this is why the column can exceed the requested `DICE_TOTAL_CFS` value.

Dataset	N	CFs/Inst	Feasible (%)	Avg L_1	Sparsity	Diversity	Time/Inst (s)
TWINS *	100	7.29 ± 3.50	55.0	16.58 ± 6.61	7.55 ± 2.05	4.14 ± 2.64	57.6 ± 12.2
COMPAS †	100	6.25 ± 3.13	96.0	8.80 ± 4.74	1.57 ± 0.48	6.49	73.5 ± 49.6
HMDA ‡	15	10.0	100	8.2×10^5	1.61	1.6×10^6	21.7

† COMPAS: N = 100; aggregates are mean ± std across successful instances. * TWINS: N = 100; feasibility 55% is immutable-`dfageq` preservation. PCA auto-enabled (22d→10d, 98% variance retained). ‡ HMDA: N = 15.

We executed EXPLAIN_COUNTERFACTUAL queries on TWINS and COMPAS, requesting three counterfactuals per instance with immutable feature constraints (`dfageq` for TWINS, `age` for COMPAS). The engine uses the R-learner as the base causal model and applies DiCE for counterfactual generation, with constraint enforcement filtering infeasible candidates before scoring. Table 17 reports counterfactual quality metrics. On COMPAS ($N = 100$), feasibility is 96.0%, with 6.25 ± 3.13 CFs/instance, $L_1 = 8.80 \pm 4.74$, diversity 6.49, and runtime 73.5 ± 49.6 s. On TWINS (N=100), feasibility is 55% under immutable `dfageq`, with 7.29 ± 3.50 CFs/instance, $L_1 = 16.58 \pm 6.61$, sparsity 7.55 ± 2.05, diversity 4.14 ± 2.64, and runtime 57.6 ± 12.2 s. HMDA uses $N = 15$.

TWINS. High-dimensional covariates ($d \approx 22$) degraded KD-tree retrieval, so the engine optionally applies PCA (auto-enabled when $d > 15$): it fits PCA on the training set, builds the KD-tree in the reduced space (22d→10d, 98% variance retained), and transforms query instances through the same PCA before nearest-neighbor search. With PCA enabled, the engine returns 7.29 ± 3.50 CFs per instance with measured L_1, diversity, and immutable feasibility (Table 17). Feasibility of 55% reflects the immutable-`dfageq` (gestational age)

constraint rather than a failed search; relaxing to monotonicity or excluding `dfageq` would be the next step for clinical utility.

7.9. Case Study: Spatial–Causal Fair Lending (HMDA)

To test expressiveness (Q1), we return to the HMDA running example (Section 4) and extend it with tract-level Census features (schema reference in Table 18). The point of this case study is to show that a spatial–causal analysis spanning multiple tables, neighborhood aggregates, and subgroup slicing is expressible as a single SQL statement. In a conventional stack, the same analysis requires five separate tools. Note: The census tract features shown here are illustrative; the actual HMDA results in Table 10 use only real HMDA fields.

Table 18. Schema reference for the HMDA spatial–causal query (columns drawn from the `hmda_loans` panel joined with FFIEC census-tract aggregates).

Column	Description
`tract_minority_population_percent`	Share of non-white residents in the tract (FFIEC Census aggregate).
`ffiec_msa_md_median_family_income`	MSA/MD median family income in USD (FFIEC reference table).
`tract_median_home_value`	Median owner-occupied home value in the tract (USD).
`tract_unemployment_rate`	Tract-level unemployment rate (proportion, 0–1).
`approval_rate`	Derived outcome: tract-level fraction of originated loans (`action_taken`$\in \{1\}$).
`debt_to_income_ratio`	Applicant DTI band reported by the lender (HMDA field).
`derived_race`	HMDA-derived race/ethnicity label used as the treatment.
`action_taken`	HMDA disposition code (1 = originated, 3 = denied, etc.); mapped into `approval_rate`.

Illustrative HMDA Query

The following query

- Joins HMDA loans with tract-level Census features;
- Computes neighborhood statistics;
- Estimates the effect of race on approval rates, controlling for applicant and neighborhood covariates.

```
WITH spatial_enrichment AS (
  SELECT h.*,
         tract_minority_population_percent,
         ffiec_msa_md_median_family_income,
         tract_median_home_value,
         tract_unemployment_rate
  FROM hmda_loans h
  JOIN census_tracts c ON h.census_tract = c.tract_code
  WHERE h.loan_amount > 0 AND h.income > 0
),
neighborhood_stats AS (
  SELECT census_tract,
         AVG(loan_amount) AS avg_loan_amount,
         COUNT(*) AS loan_count,
         SUM(CASE WHEN action_taken = 1 THEN 1 ELSE 0 END) AS approvals
  FROM spatial_enrichment
  GROUP BY census_tract
)
SELECT EXPLAIN_CAUSALLY_WHY approval_rate,
       ADJUSTING (income, loan_amount, debt_to_income_ratio,
                  tract_minority_population_percent,
                  ffiec_msa_md_median_family_income,
                  tract_median_home_value,
                  tract_unemployment_rate),
       INTERVENE_WITH derived_race,
       USING_MODEL meta_learners
       WITH_PARAMETERS (
         RETURN_CI=1,
         DIAGNOSTICS=1,
         SAMPLE_SIZE=5000,
         CROSS_FIT_FOLDS=3
```

```
    )
FROM spatial_enrichment s
JOIN neighborhood_stats n ON s.census_tract = n.census_tract
WHERE s.derived_race IN ('White', 'Black or African American', 'Hispanic or Latino')
  AND s.loan_amount BETWEEN 50000 AND 2000000
ORDER BY s.census_tract, s.income DESC
LIMIT 5000;
```

In a conventional stack, this analysis requires:

- SQL for the initial HMDA filters;
- A spatial join in Python/R (GeoPandas [67] or `sf` [68]);
- Preparation of neighborhood aggregates;
- A CausalML or custom causal script;
- Ad hoc glue to merge diagnostics back to the database.

The output is a relational table of treatment effects and diagnostics keyed by tract and subgroup.

Identification note. With `DIAGNOSTICS = 1`, the HMDA run reports max SMD $= 0.190$, indicating moderate covariate imbalance; analysts should apply overlap enforcement (e.g., propensity score trimming to $[0.1, 0.9]$) before interpreting treatment effects.

7.10. Case Study: Census Employment Disparities

We next apply the system to a Census-based employment analysis, asking the following:

> Does race affect employment rates after controlling for education, income, age, and other demographics?

We construct a synthetic sample matching ACS 2022 microdata characteristics (schema reference in Table 19; column names, distributions, and sample size from ACS PUMS documentation), focusing on working-age adults and a subset of racial/ethnic groups, and control for human capital, economic status, and geography.

Table 19. Schema reference for the Census employment query (ACS PUMS variable names retained; see the Census Bureau PUMS Data Dictionary [63] for encodings).

Column	Description
`PINCP`	Person's total income in past 12 months (USD, PUMS).
`AGEP`	Age in years (PUMS).
`SCHL`	Educational attainment code (PUMS, ordinal).
`SEX`	Biological sex code (1 = male, 2 = female; PUMS).
`MAR`	Marital status code (PUMS, 5 levels).
`NATIVITY`	Native-born vs. foreign-born indicator (PUMS).
`PUMA`	Public Use Microdata Area (geographic unit, $\geq$100k population).
`DIVISION`	Census Division code (1–9, PUMS).
`income_log`	Derived: $\log(1 + \texttt{PINCP})$ for heavy-tail correction.
`age_squared`	Derived: $\texttt{AGEP}^2$, standard Mincer-style quadratic term.
`has_health_insurance`	Derived binary: any coverage flag set in `HICOV`/private/public PUMS fields.
`race_ethnicity`	Derived categorical: collapsed from `RAC1P`/`HISP` into White/Hispanic-or-Latino/Other.
`is_white`	Derived binary treatment: 1 iff `race_ethnicity` = White.
`employed`	Derived binary outcome: 1 iff `ESR`$\in \{1, 2\}$ (civilian employed at work or with a job).

7.10.1. Illustrative Census Query

The analysis is expressed as a single EXPLAIN_CAUSALLY_WHY query:

```
SELECT EXPLAIN_CAUSALLY_WHY employed,
      ADJUSTING (
          PINCP, AGEP, SCHL,
          SEX, MAR, NATIVITY,
          PUMA, DIVISION,
          income_log, age_squared, has_health_insurance
      ),
```

```
        INTERVENE_WITH is_white,
        USING_MODEL auto_ensemble
        WITH_PARAMETERS (
            MODELS='slearner,tlearner,xlearner,drlearner',
            RETURN_MODEL_BREAKDOWN=1,
            RETURN_STABILITY_INDEX=1,
            RETURN_CI=1,
            DIAGNOSTICS=1,
            FAIRNESS_SLICES='race_ethnicity',
            SAMPLE_SIZE=8000,
            CROSS_FIT_FOLDS=3,
            POSITIVITY_BOUNDS='0.1:0.9',
            n_estimators=50,
            max_depth=6,
            learning_rate=0.1
        )
FROM census_data
WHERE PINCP > 0 AND PINCP < 200000
  AND AGEP BETWEEN 18 AND 65
  AND race_ethnicity IN ('White', 'Hispanic or Latino', 'Other')
ORDER BY PINCP DESC, AGEP DESC
LIMIT 5000;
```

`POSITIVITY_BOUNDS` is honored for overlap diagnostics when `DIAGNOSTICS=1`.

7.10.2. Results

The synthetic ACS-like sample (auto-ensemble; ATE 0.115, max SMD 0.031, $N = 9423$) demonstrates pipeline behavior under good balance and should not be extrapolated to real employment disparities, whereas the real ACS PUMS run (R-learner; ATE 0.005, 95% CI $[-0.013, 0.013]$ crossing zero, max SMD 0.377, $N = 10{,}000$) correctly flags unconfoundedness/positivity failure under severe imbalance.

7.11. Summary

On expressiveness (Q1), the HMDA spatial–causal case study (Section 7.9) replaces a five-tool external pipeline with one SQL statement. Multi-table joins, fairness slices, and counterfactuals with constraint enforcement are all expressible in the dialect. On statistical quality (Q2), the results are mixed: the ψ operator recovers a published protective ATE on TWINS (-0.103, consistent with [17,62]) and returns a non-significant COMPAS point ATE (-0.045; CI crosses zero; max SMD $= 0.335$), which we treat as indicative only rather than a detected racial disparity; HMDA is correctly flagged as an identification failure (max SMD > 0.10). Census (synthetic) is a balanced synthetic benchmark (good balance, ATE 0.115); the real ACS diagnostic run (R-learner) gives ATE 0.005 with max SMD 0.377, so the real row illustrates diagnostic flagging under imbalance rather than a policy-grade employment effect. Counterfactual quality is reported for COMPAS and TWINS at $N = 100$ (Table 17); the dual-index comparison at $N = 100$ appears in Table 12. On systems behavior (Q3), COMPAS per-method runtimes span ≈ 0.2–11 s, but full-sample TWINS ($\approx$3232 s) and Census ($\approx$475 s) end-to-end runs are far from interactive (Section 7.7). While full TWINS takes $\approx$3232 s—model training accounts for 85–99% of wall-clock time, and SQL parsing overhead is negligible (<0.1 s).

8. Limitations and Future Work

All reported ATEs are observational: they are valid causal effects only under unconfoundedness, positivity, consistency, and SUTVA, none of which the engine can certify from data alone. Relational pushdown and rewrite locality improve execution engineering but do not relax those identification assumptions. The system surfaces the assumptions through the ADJUSTING and INTERVENE_WITH clauses and, when `DIAGNOSTICS = 1` is enabled, flags poor propensity overlap and covariate imbalance via SMD; making the checks visible inside SQL is useful, but an SMD warning or a returned CI does not convert

an exploratory disparity or a fairness diagnostic into a policy-grade causal claim. Certifying that identification assumptions hold—and choosing the appropriate claim tier in Table 9—remains the analyst's job.

8.1. Threats to Validity

We briefly restate the main validity concerns here. Internal validity. All evaluation datasets are observational, so ATEs are credible only under unconfoundedness and positivity; COMPAS and HMDA already show substantial or moderate covariate imbalance (max SMD > 0.10), and unobserved confounders may remain. External validity. Effects estimated on California HMDA slices, Broward County COMPAS, and 1989 TWINS birth records may not generalize to other jurisdictions or eras. Construct validity. Treating race as a binary treatment (`is_white`) detects disparities under the model's assumptions, not ground-truth discrimination.

We place extended SQL surface listings and advanced composition examples in Appendix A, hyperparameter tables and connecting-the-steps implementation walkthroughs in Appendix B, and extended experimental material (ZaliQL head-to-head detail, CATE tables, scalability plots, and dual-index ablation) in Appendix C, so the main text focuses on the core operators, system organization, and headline evaluation findings.

The counterfactual evaluation covers COMPAS ($N = 100$, 96.0% feasibility) and TWINS ($N = 100$, 55% immutable-`dfageq` feasibility with L_1/diversity metrics; Table 17). The dual-index ablation at $N = 100$ with diversity and bootstrap CIs is reported in Table 12. We have not tested on additional clinical or financial recourse domains where feasibility constraints are most complex; the TWINS 55% feasibility rate is itself a reminder that hard immutability constraints can dominate CF yield.

Expressiveness (Q1) is argued by representative query examples rather than controlled experiments with practitioners. Broadening the counterfactual evaluation and validating expressiveness with domain experts are prerequisites for any claim of generality.

On the methods side, we would like to add instrumental variables [24,32], difference-in-differences [69], and sensitivity analysis (E-values [70], Rosenbaum bounds [71]) as declarative SQL clauses. This paper specifies the operational semantics of ψ and φ in Section 4 (schemas, treatment typing, rewrite Rules 1–4, and the validity/admissibility bridge); a full denotational semantics in potential outcomes and relational algebra remains future work.

8.2. Bootstrap and Inferential Limits

For compute reasons, the default bootstrap is $B = 50$ iterations on TWINS, COMPAS, and synthetic Census, and $B = 10$ on HMDA (reported as a 10-iteration min/max spread rather than a 95% CI). Standard practice is $B \geq 1000$ for tight percentile or BCa intervals; at our sample sizes, a $B \geq 500$ pass on HMDA would require ≈ 32 h on the target hardware and is deferred to a follow-up study with cluster compute. The point estimates and the direction/ordering of effects remain the primary evidence; readers should treat the CIs as indicative.

Finally, there is the risk of epistemic overconfidence. A single SQL statement that returns "ATE $= -0.045$" looks authoritative in a way that a 200-line Python notebook does not. Someone unfamiliar with observational causal inference could easily report that number as "the causal effect of race on recidivism" without understanding the unconfoundedness assumption, the SMD caveats, or the fact that the confidence interval includes zero. The built-in diagnostics are designed to resist this, as they surface SMD warnings and CI bounds directly in the output table, but diagnostics can only help analysts if they read them.

9. Conclusions

We have introduced `EXPLAIN_CAUSALLY_WHY` and `EXPLAIN_COUNTERFACTUAL` as SQL-dialect operators, together with an end-to-end system PsiQL that translates relational query plans into causal specifications and executes the query plan. PsiQL integrates meta-learners, cross-fitting, diagnostics, and dual-index counterfactual search, returning table-valued results. Evaluation shows that realistic fairness and clinical analyses are expressible as single SQL queries; that COMPAS yields a non-significant point ATE whose CI crosses zero and is best read as an exploratory disparity under claim tiers (Table 9); that synthetic Census is a balanced pipeline proof-of-concept while real ACS remains fairness-diagnostic-only under severe imbalance; and that the primary contribution is a composable SQL-dialect surface for meta-learner ensembles, diagnostics, and constraint-respecting counterfactuals rather than uniformly faster runtime than classical SQL estimators. Detailed numbers and caveats appear in Section 7; limitations and threats to validity are discussed in Section 8.

Author Contributions: Conceptualization, R.P. and S.R.; methodology, R.P. and S.R.; software, R.P.; validation, R.P., S.R. and M.A.A.; formal analysis, R.P.; investigation, R.P.; data curation, R.P.; writing—original draft preparation, R.P.; writing—review and editing, R.P., S.R. and M.A.A.; visualization, R.P.; supervision, S.R. and M.A.A.; project administration, S.R. and M.A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the New Brunswick Innovation Foundation (NBIF), grant number AIP_2023_002.

Data Availability Statement: All datasets used in this study are publicly available: HMDA from the FFIEC/CFPB at https://ffiec.cfpb.gov/data-publication/ (accessed on 3 August 2026); COMPAS from ProPublica at https://github.com/propublica/compas-analysis (accessed on 3 August 2026); TWINS from the standard benchmark distribution of Almond, Chay, and Lee [62]; and ACS PUMS from the U.S. Census Bureau at https://www.census.gov/programs-surveys/acs/microdata.html (accessed on 3 August 2026). The PsiQL prototype source code and evaluation scripts used to reproduce the experiments will be made available by the authors on request.

Acknowledgments: The authors thank the University of New Brunswick and the Université de Moncton for resources used in this work. During the preparation of the manuscript, the authors used Cursor 3.15.6 (Anysphere) to support LaTeX drafting and refactoring. The authors have reviewed and edited all output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. SQL Queries Extended Examples

This appendix consolidates the generic SQL queries referenced from Section 4.5 and Section 4.8. Clause semantics for `ADJUSTING`, `INTERVENE_WITH`, `USING_MODEL`, `WITH_PARAMETERS`, and Rule 4 `GROUP BY` (post-ψ stratum loop; unsupported for φ) appear in the main text; generic ψ/φ templates and CTE composition patterns are consolidated here as a single syntax reference.

Generic ψ template.

```
SELECT EXPLAIN_CAUSALLY_WHY outcome_expr,
       ADJUSTING (covariate_1, ..., covariate_k),
       INTERVENE_WITH treatment_expr,
       [USING_MODEL model_name],
       [WITH_PARAMETERS (param_key = value, ...)]
FROM ... [JOIN ...] [WHERE ...] [GROUP BY ...];
```

Generic φ template (preferred surface).

```
SELECT * FROM EXPLAIN_COUNTERFACTUAL(
  (SELECT ... FROM ... [JOIN ...] [WHERE ...])
  OUTCOME outcome_col TARGET target_value
  TOTAL_CFS k IMMUTABLE col_1 [, col_2, ...]
);
```

Advanced composition. CTEs and subqueries in `FROM` prepare cohorts; operators appear at top-level `SELECT`. Window functions, `CASE`, `GROUP BY`/`HAVING` (for ψ only, Rule 4), and `ORDER BY` compose with downstream `JOIN`s on materialized ψ/φ output relations.

Appendix B. Hyperparameters and Implementation Walkthroughs

Table A1 lists every hyperparameter governing the evaluation pipeline. Algorithm walkthroughs for ψ (Rule 4 `GROUP BY` loop at lines 25–27) and φ (four-phase dual-index pipeline) complement the discussion before Algorithm 1.

Table A1. Hyperparameters of the PsiQL causal and counterfactual pipelines used throughout the evaluation.

Component	Parameter	Value	Notes
XGBoost base learner	`n_estimators`	100	library default
XGBoost base learner	`max_depth`	6	library default
XGBoost base learner	`learning_rate`	0.1	library default
Meta-learners	K-fold CV (R/DR)	$K = 5$	Chernozhukov-style cross-fit [31]
Bootstrap	iterations (TWINS/ COMPAS/Census)	$B = 50$	compute-bound; see Section 8
Bootstrap	iterations (HMDA)	$B = 10$	reported as min/max spread, not 95% CI
Bootstrap	CI type	percentile	
SMD	imbalance threshold	0.10	Rosenbaum/Stuart convention [26]
DiCE	method	random	DiCE v0.11 [35]
DiCE	`total_CFs`	3 (COMPAS/TWINS), 10 (HMDA)	per-instance request
DiCE	`proximity_weight`	0.5	library default
DiCE	`diversity_weight`	1.0	library default
Dual index	`USE_KDETREE`	`true`	shipped default (diversity–proximity trade-off; not quality-dominant)
Dual index	`USE_LSH`	`true`	shipped default (diversity–proximity trade-off; not quality-dominant)
KD-tree	`leaf_size`	40	scikit-learn default
MinHash LSH	`num_perm` (m)	128	standard
LSH	Jaccard threshold	0.5	
PCA	auto-enable dimension	$d > 15$	engineered heuristic
PCA	variance retained	98%	
Seed	NumPy, XGBoost, sklearn, DiCE	42	all RNGs seeded

Algorithm 1 walkthrough. Lines 1–4 parse and bind; lines 5–15 dispatch meta-learners; lines 16–24 add cross-fit, bootstrap CIs, and SMD diagnostics; lines 25–27 apply Rule 4 (`GROUP BY` $\Rightarrow$ one ψ call per stratum, union rows).

Algorithm 2 walkthrough. Phase 1 builds KD-tree and LSH indices (respecting `USE_KDETREE`/`USE_LSH`); Phase 2 unions neighbor pools; Phase 3 runs DiCE; Phase 4 filters constraints and ranks by proximity/diversity.

Appendix C. Extended Experimental Results

Additional evaluation material cited in Section 7 includes the ZaliQL comparison (Table 13), CATE subgroups (Tables 14 and 15), dual-index ablations (Tables 11 and 12), counterfactual quality (Table 17), and scalability results (Figure 6, Table 16).

References

1. Chamberlin, D.D.; Boyce, R.F. SEQUEL: A structured English query language. In *Proceedings of 1974 ACM SIGFIDET Workshop on Data Description, Access and Control*; ACM: New York, NY, USA,1974; pp. 249–264. [CrossRef]

2. Obermeyer, Z.; Powers, B.; Vogeli, C.; Mullainathan, S. Dissecting racial bias in an algorithm used to manage the health of populations. *Science* **2019**, *366*, 447–453. [CrossRef] [PubMed]
3. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. *Adv. Neural Inf. Process. Syst.* **2017**, *30*. [CrossRef]
4. Ribeiro, M.T.; Singh, S.; Guestrin, C. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 2016; pp. 1135–1144.
5. Slack, D.; Hilgard, S.; Jia, E.; Singh, S.; Lakkaraju, H. Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*; ACM: New York, NY, USA, 2020; pp. 180–186.
6. Huang, X.; Marques-Silva, J. On the Failings of Shapley Values for Explainability. *Int. J. Approx. Reason.* **2024**, *159*, 109–112. [CrossRef]
7. Lipton, Z.C. The mythos of model interpretability. *Queue* **2018**, *16*, 31–57. [CrossRef]
8. Chen, H.; Harinen, T.; Lee, J.Y.; Yung, M.; Zhao, Z. CausalML: Python package for causal machine learning. *arXiv* **2020**, arXiv:2002.11631.
9. Battocchi, K.; Dillon, E.; Hei, M.; Lewis, G.; Oka, P.; Oprescu, M.; Syrgkanis, V. EconML: A Python Package for ML-Based Heterogeneous Treatment Effects Estimation. In *Proceedings of the Microsoft Research Technical Report*; Version 0.x.; Microsoft Corporation: Redmond, WA, USA, 2019.
10. Sharma, A.; Kiciman, E. DoWhy: An end-to-end library for causal inference. *arXiv* **2020**, arXiv:2011.04216.
11. Salimi, B.; Suciu, D. ZaliQL: A SQL-Based Framework for Drawing Causal Inference from Big Data. *arXiv* **2016**, arXiv:1609.03540.
12. Salimi, B.; Cole, C.; Ports, D.R.; Suciu, D. ZaliQL: Causal Inference from Observational Data at Scale. *Proc. Vldb Endow.* **2017**, *10*, 1957–1960.
13. Salimi, B.; Parikh, H.; Kayali, M.; Roy, S.; Getoor, L.; Suciu, D. Causal Relational Learning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 2020; pp. 241–256.
14. Hellerstein, J.; Ré, C.; Schoppmann, F.; Wang, D.Z.; Fratkin, E.; Gorajek, A.; Ng, K.S.; Welton, C.; Feng, X.; Li, K.; et al. The MADlib Analytics Library or MAD Skills, the SQL. *arXiv* **2012**, arXiv:1208.4165.
15. Cox, D.R. *Planning of Experiments*; John Wiley & Sons, Inc.: New York, NY, USA; Chapman & Hall, Ltd.: London, UK, 1958; pp. vii + 308.
16. Peter, R. SQL with Causal Inference and Counterfactual Reasoning for Explainable Analytics. Master's Thesis, University of New Brunswick, Fredericton, NB, Canada, 2025. Available online: https://unbscholar.lib.unb.ca/items/0ba415ff-016b-47e5-9444-379cd10c03f8 (accessed on 3 August 2026).
17. Künzel, S.R.; Sekhon, J.S.; Bickel, P.J.; Yu, B. Metalearners for estimating heterogeneous treatment effects using machine learning. *Proc. Natl. Acad. Sci. USA* **2019**, *116*, 4156–4165. [CrossRef] [PubMed]
18. Federal Financial Institutions Examination Council (FFIEC); Consumer Financial Protection Bureau (CFPB). Home Mortgage Disclosure Act (HMDA) Public Data Platform. 2023. Available online: https://ffiec.cfpb.gov/data-publication/ (accessed on 3 August 2026).
19. Pearl, J. *Causality: Models, Reasoning, and Inference*, 2nd ed.; Cambridge University Press: Cambridge, UK, 2009.
20. Pearl, J.; Mackenzie, D. *The Book of Why: The New Science of Cause and Effect*; Basic Books: New York, NY, USA, 2018.
21. Shmueli, G. To Explain or to Predict? *Stat. Sci.* **2010**, *25*, 289–310. [CrossRef]
22. Holland, P.W. Statistics and Causal Inference. *J. Am. Stat. Assoc.* **1986**, *81*, 945–960. [CrossRef]
23. Rubin, D.B. Estimating causal effects of treatments in randomized and nonrandomized studies. *J. Educ. Psychol.* **1974**, *66*, 688–701. [CrossRef]
24. Imbens, G.W.; Rubin, D.B. *Causal Inference in Statistics, Social, and Biomedical Sciences*; Cambridge University Press: Cambridge, UK, 2015.
25. Rosenbaum, P.R.; Rubin, D.B. The Central Role of the Propensity Score in Observational Studies for Causal Effects. *Biometrika* **1983**, *70*, 41–55. [CrossRef]
26. Stuart, E.A. Matching methods for causal inference: A review and a look forward. *Stat. Sci.* **2010**, *25*, 1–21. [CrossRef] [PubMed]
27. Kang, J.D.Y.; Schafer, J.L. Demystifying double robustness: A comparison of alternative strategies for estimating a population mean from incomplete data. *Stat. Sci.* **2007**, *22*, 523–539. [CrossRef] [PubMed]
28. Nie, X.; Wager, S. Quasi-oracle estimation of heterogeneous treatment effects. *Biometrika* **2021**, *108*, 299–319. [CrossRef]
29. Kennedy, E.H. Towards optimal doubly robust estimation of heterogeneous causal effects. *Electron. J. Stat.* **2023**, *17*, 3008–3049. [CrossRef]
30. Athey, S.; Wager, S. Estimating treatment effects with causal forests: An application. *Obs. Stud.* **2019**, *5*, 37–51. [CrossRef]
31. Chernozhukov, V.; Chetverikov, D.; Demirer, M.; Duflo, E.; Hansen, C.; Newey, W.; Robins, J. Double/debiased machine learning for treatment and structural parameters. *Econom. J.* **2018**, *21*, C1–C68. [CrossRef]
32. Angrist, J.D.; Imbens, G.W.; Rubin, D.B. Identification of Causal Effects Using Instrumental Variables. *J. Am. Stat. Assoc.* **1996**, *91*, 444–455. [CrossRef]

33. Wachter, S.; Mittelstadt, B.; Russell, C. Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR. *Harv. J. Law Technol.* **2017**, *31*, 842–887.
34. Mittelstadt, B.; Russell, C.; Wachter, S. Explaining Explanations in AI. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 279–288.
35. Mothilal, R.K.; Sharma, A.; Tan, C. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*; ACM: New York, NY, USA, 2020; pp. 607–617.
36. Joshi, S.; Koyejo, O.; Vijitbenjaronk, W.; Kim, B.; Ghosh, J. Towards Realistic Individual Recourse and Actionable Explanations in Black-Box Decision Making Systems. In Proceedings of the NeurIPS Workshop on Robust AI in Financial Services, Vancouver, BC, Canada, 13–14 December 2019.
37. Ustun, B.; Spangher, A.; Liu, Y. Actionable Recourse in Linear Classification. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 10–19.
38. Prosperi, M.; Guo, Y.; Sperrin, M.; Koopman, J.; Min, J.; He, X.; Rich, S.; Wang, M.; Buchan, I.; Bian, J. Causal inference and counterfactual prediction in machine learning for actionable healthcare. *Nat. Mach. Intell.* **2020**, *2*, 369–375. [CrossRef]
39. Bentley, J.L. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* **1975**, *18*, 509–517. [CrossRef]
40. Broder, A.Z. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences*; IEEE: Piscataway, NJ, USA, 1997; pp. 21–29.
41. Indyk, P.; Motwani, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*; ACM: New York, NY, USA, 1998; pp. 604–613.
42. Sundaram, N.; Turmukhametova, A.; Satish, N.; Mostak, T.; Indyk, P.; Madden, S.; Dubey, P. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *Proc. Vldb Endow.* **2013**, *6*, 1930–1941. [CrossRef]
43. Roy, S.; Suciu, D. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 2014; pp. 507–518.
44. Meliou, A.; Gatterbauer, W.; Halpern, J.Y.; Koch, C.; Moore, K.F.; Suciu, D. Causality in Databases. *IEEE Data Eng. Bull.* **2010**, *33*, 59–67.
45. Eisenberg, A.; Melton, J. Standards: SQL: 1999, formerly known as SQL3. *Sigmod Rec.* **1999**, *28*, 131–138.
46. Mumick, I.S.; Pirahesh, H. Implementation of Magic-sets in a Starburst Extensible Database System. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 1994; pp. 103–114. [CrossRef]
47. Raasveldt, M.; Mühleisen, H. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*; ACM: New York, NY, USA, 2019; pp. 1981–1984.
48. Watson, A.; Das, S.K.; Ray, S. DaskDB: Scalable Data Science with Unified Data Analytics and In Situ Query Processing. In *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (DSAA)*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–10.
49. Feng, X.; Kumar, A.; Recht, B.; Ré, C. Towards a unified architecture for in-RDBMS analytics. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*; ACM: New York, NY, USA, 2012; pp. 325–336.
50. Karanasos, K.; Interlandi, M.; Xin, D.; Psallidas, F.; Sen, R.; Park, K.; Popivanov, I.; Nakandal, S.; Krishnan, S.; Weimer, M.; et al. Extending relational query processing with ML inference. *arXiv* **2019**, arXiv:1911.00231.
51. Makrynioti, N.; Ley-Wild, R.; Vassalos, V. sql4ml: A declarative end-to-end workflow for machine learning. *arXiv* **2019**, arXiv:1907.12415.
52. Google Cloud. BigQuery ML: Create and Execute Machine Learning Models Using Standard SQL Queries. 2024. Available online: https://cloud.google.com/bigquery-ml/docs/introduction (accessed on 3 August 2026).
53. SAP SE. SAP HANA Predictive Analysis Library (PAL) Reference. 2024. Available online: https://help.sap.com/docs/SAP_HANA_PLATFORM/2cfbc5cf2bc14f028cfbe2a2bba60a50/f652a8186a144e929a1ade7a3cb7abe8.html (accessed on 3 August 2026).
54. Microsoft. SQL Server Machine Learning Services with R and Python. 2024. Available online: https://learn.microsoft.com/en-us/sql/machine-learning/sql-server-machine-learning-services (accessed on 3 August 2026).
55. Robinson, P.M. Root-*N*-Consistent Semiparametric Regression. *Econometrica* **1988**, *56*, 931–954. [CrossRef]
56. Mao, T. SQLGlot: Python SQL Parser and Transpiler. 2024. Available online: https://github.com/tobymao/sqlglot (accessed on 3 August 2026).
57. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 2016; pp. 785–794.
58. McKinney, W. Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference, Austin, TX, USA, 28 June–3 July 2010; pp. 56–61.
59. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
60. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef] [PubMed]

61. Angwin, J.; Larson, J.; Mattu, S.; Kirchner, L. *Machine Bias*; ProPublica: New York, NY, USA, 2016.
62. Almond, D.; Chay, K.Y.; Lee, D.S. The costs of low birth weight. *Q. J. Econ.* **2005**, *120*, 1031–1083. [CrossRef]
63. U.S. Census Bureau. American Community Survey Public Use Microdata Sample (PUMS). 2023. Available online: https://www.census.gov/programs-surveys/acs/microdata.html (accessed on 3 August 2026).
64. Dressel, J.; Farid, H. The accuracy, fairness, and limits of predicting recidivism. *Sci. Adv.* **2018**, *4*, eaao5580. [CrossRef] [PubMed]
65. Kohler-Hausmann, I. Eddie Murphy and the Dangers of Counterfactual Causal Thinking About Detecting Racial Discrimination. *Northwestern Univ. Law Rev.* **2018**, *113*, 1163–1228.
66. Crump, R.K.; Hotz, V.J.; Imbens, G.W.; Mitnik, O.A. Dealing with Limited Overlap in Estimation of Average Treatment Effects. *Biometrika* **2009**, *96*, 187–199. [CrossRef]
67. Jordahl, K.; Van den Bossche, J.; Fleischmann, M.; Wasserman, J.; McBride, J.; Gerard, J.; Tratner, J.; Perry, M.; Badaracco, A.G.; Farmer, C.; et al. *Geopandas/Geopandas: v0.8.1*; Zenodo: Geneva, Switzerland, 2020. [CrossRef]
68. Pebesma, E. Simple Features for R: Standardized Support for Spatial Vector Data. *R J.* **2018**, *10*, 439–446. [CrossRef]
69. Angrist, J.D.; Pischke, J.S. *Mostly Harmless Econometrics: An Empiricist's Companion*; Princeton University Press: Princeton, NJ, USA, 2009.
70. VanderWeele, T.J.; Ding, P. Sensitivity Analysis in Observational Research: Introducing the E-Value. *Ann. Intern. Med.* **2017**, *167*, 268–274. [CrossRef] [PubMed]
71. Rosenbaum, P.R. *Observational Studies*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2002.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.