Shreekant Malviya[1]
Abhishek Jaiswal[2]
Vivek Koli[3]

A DBT-Based Column Lineage Tracking Approach for Regulatory and Audit Compliance

Journal of Electrical Systems

***Abstract*:** Regulatory programs are also becoming more and more intuitive about demonstrated column-level lineage of PII/PCI/PHI and financial qualities, which many organizations now have only table-based DAGs. The paper presents a compile-time solution based on the dbt-first proposal, currently processing compiled SQL and dbt artifacts to produce normalized column-to-column edges and storing them in an open governance store, which is mapped to control evidence. The strategy was tested with a production-similar project (≈250 models, 1,800 columns, ~9,000 edges) and Snowflake and BigQuery profiles in 1,200- 1,800-day schedules of tasks. The accuracy was measured on a stratified, dual-view ground truth (n=400): precision 0.971 (95% CI 0.952–0.987), recall 0.934 (0.905–0.960), F1 0.952. There was coverage of 96.7 overall (customer 97.8%, payments 96.4%, and health 95.1%) fields tagged with critical coverage. Operation impact was limited: compile/run deltas of between 7.6% and lineage metadata of less than 5 GB/month, and platform cost deltas of less than 250/month at scale stated. The compliance results were significantly improved: median DSAR turnaround decreased by 12.1 to 4.3 days (−64%); SOX evidence-pack assembly decreased to 18 minutes, and quarterly defect leakage remained 1.6%. Recall gains were seen in ablations with materialization of transient models (+3.2 pp), macro-depth limits and explicit select lists (+2.9 pp), and propagation of the best-effort UDF (+1.7 pp). Dynamic SQL, non-dbt pipelines are also limited; the key areas of future development are sub-minute streaming lineage, semantic equivalence +37 pp recall, and interoperability using standards.

***Keywords*:** dbt-based column lineage, Regulatory & audit compliance, PII/PCI/PHI data governance, OpenLineage / Apache Atlas integration, DSAR (Data Subject Access Requests).

I. INTRODUCTION

Present-day analytics packages are becoming widely transformed to use dbt to coordinate SQL constructs throughout a cloud data warehouse. With growing model inventories, table-level lineage is no longer viable to meet compliance use cases that demand column-level traceability of personally identifiable information (PII), credit card information (PCI), and protected health information (PHI). In a mid-size enterprises, 200-500 dbt models normally generate 1500-6000 number of columns and 10000- 40,000 number of column to column edges. Through manual steps, a single field of sensitivity may need 3-8 analyst hours to trace through this graph, and it is amplified with repeated requests. A feasible dbt-native system that derives columnage in normal development has guaranteed a level of accuracy that easily meets measurements, reduced the cost of compliance, prompt audit reaction, and has minimal impact on delivery schedules.

The regulatory regimes focus on provable traceability and the effectiveness of control. The processing activities of Article 30 of GDPR presuppose the records addressing where and how attributes of the personal data, such as customer_email or national_id, are transformed and stored. SOX 404 anticipates controls evidence that operates the financial reporting controls, such as proof of derivative measures balancing to the managed sources and changes being authorized and tested. HIPAA 164.312 requires technical safeguarding of ePHI, and therefore, propagation of PHI tags via transformations is crucial. Auditors periodically demand the lineage to show me on a limited number of columns, and evidence packs regularly contain 50-100 artifacts in SQL, code reviews, run logs, and approvals. The number of 2-10 data subject access requests (DSARs) per 10,000 customers/month is another typical feature of many organizations; with no column lineage, weeks could easily turn into days when responding. These requirements bring about a tangible requirement of accuracy and verifiable column lineage in the operational development processes.

The main issue of discussion is whether a dbt-based compile-time parsing solution can generate the correct continuous column lineage at an exclusionary cost. The main aim is to attain 95% accuracy at the column edge level and maintain a 90 percent recall on an average dbt project. The second goal is to measure the overhead of operation with ambitious aims of compile and run-time deltas of 10% or less and less than 5 GB of lineage metadata. The third goal is to determine the compliance impact and examine whether the evidence-based automated lineage is able to decrease the DSAR and audit cycle time by 50-70 percent and reduce the labor needed to assemble evidence by 80-120 hours per quarter. These goals, together, make the evaluation based on statistically significant results that can be checked by the practitioners through a reproducible sample, baseline comparison, and confidence interval.

This study has various contributions as it proposes a dbt-first system that compiles the SQL and dbt objects, needs edges based on common table expressions, subqueries, macro expansions, label expansions, and short-lived models. It establishes a measurement scheme, edge-level quality, recall, F1, coverage of tagged sensitive fields, p95 compile and run-time, storage and cost deltas, associated with audit results, including cycle time and defect leakage. The study also traces the lineage outputs to governance checks, which allow reproducible evidence

identifiable to tracking of processing, change management, and access review demands. It offers a reproducible assessment model based on stratified samples of 300-500 edges in dual-review labeling and Cohen's $\kappa \geq 0.80$ for agreement.

The scopes of the study include projects of DBT (v1.5+) written in Snowflake or BigQuery that are limited to SQL transformations that are compilable through ref, source, and exposure constructs. Some of the exclusions are non-dbt pipelines, runtime black-box transformations, cross-platform orchestration out of dbt, or even the entire user-defined functions semantics beyond name-level propagation. Utilized types of attention are practical focus on automated export of the dbt target folder and enduring into an open lineage database and CI/CD assimilated deployments, regression catch, and standardized evidence creation.

This study is structured into different chapters. Chapter 2 offers a critical literature review and surveys compliance tooling applicable in column-level lineage, and identifies gaps that the study fills. Chapter 3 presents the research design, such as datasets, dbt-based extraction pipeline, evaluation metrics, and governance mapping, as well as the methods of analysis. Chapter 4 presents empirical results, in terms of accuracy and coverage rates, performance and cost overheads, compliance rates, and ablation and sensitivity analyses to verify robustness. Chapter 5 interprets these findings, covering operational trade-offs, limitations, validity threats, and implications on balanced data management and readiness of audit affairs. Chapter 6 provides research directions for the future, including real-time lineage, semantic enrichment, and standards interoperability. The study concludes with the synthesis of contributions to the study, practical recommendations for adoption to be followed, and the much larger implications of the research to regulatory and auditing compliance.

II. LITERATURE REVIEW

2.1 Foundations of lineage & provenance

The data lineage refers to the flow of data attributes through data transformation between outputs and sources. Graphically, nodes are models, tables, and columns, and directed edges are derivations, provenance adds context to such a graph on execution-attributes, namely who changed what and when, and with which configuration, to enable auditors to follow decision paths. Column-level lineage plays a critical role in a controlled environment since one attribute, such as customer-email or card-token, can be used dozens of times in a join, cast, or aggregation until it finally shows up in reporting marts.

There are two complementary modes of capture. There is also the case of static lineage, which derives the best edges on code or compiled SQL, which provides determinism, reproducibility, and low runtime overhead [1]. Runtime lineage deduces meshings out of executed query designs and tracking and recording tardiness characteristics of conditional logic and portions of user-proscribed functions. Warehouse-sized dbt projects will often include 200-500 models, 1500-6000 columns, and 10,000-20,000 column-column edges; by that point, manual tracking is no longer feasible and results in bias. Practically, lineage nodes need to store metadata sensitivity tags, their owners, SLAs, and test status so that downstream risk analysis, access reviews, and evidence generation may be automated and quantified [2].

2.2 Tooling survey

dbt generates machine-readable artifacts: manifest.json, catalog.json, and run results.json, which represent SQL of model queries, model schemas, model tests, macros, and the results of model execution. By assembling compiled SQL in the target/ directory, responding to ref() and source () calls, normalizing the reality of imaginary names, and generating edges of the form (source_model.column → target_model.column). OpenLineage/Marquez and Apache Atlas keep nodes and edges, as well as run metadata; Databricks Unity Catalog discloses lineage in the lakehouse. Cloud observability operational practice is educational: successful operations programs include centralization of high cardinality telemetry and mapping causal paths across systems, coupled with defining service level goals.

As shown in Figure 1 below, the image depicts the working process and artefacts in the process of compiling and testing a model by dbt. The left side shows the dbt project directory structure, with several important folders, including some of the key ones, like models, snapshots, and target, in which the compiled SQL files and other artifacts related to a lineage, such as manifest.json, run-results.json, and graph-summary.json, are stored. These objects are the basis of building lineage graphs through the resolution of ref and source relationships in SQL models [3]. The right-hand side represents an overview of dbt monitoring dashboard with test results, health of tables, and models executed, with the metrics being test pass rates, schema changes, anomalies, and decays. These visual aspects, as a group, highlight the infrastructure of the data lineage ecosystem, such as between model compilation and metrics of the execution, as it allows teams to trace their dependencies, validate the transformations, and have an observability of all models of the warehouse, which wants to be compliant.

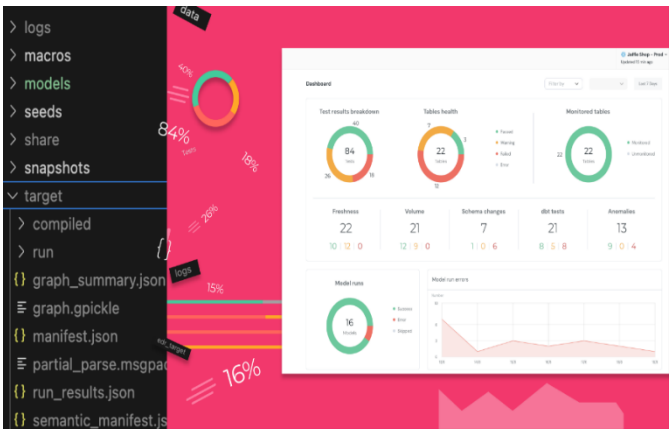

Figure 1: dbt artifacts and dashboard for compiled SQL lineage tracking.

When applied by analogy, the extraction of lineage ought to be real-time, cover import metrics and progress, and interoperate with alerting in such a way that materialized views with sensitive information should be unable to evolve unless similarly synchronized lineage is updated. Commonly occurring engineering holes in macro expansion, ephemeral models, and dynamic SQL, which rewrites the select list at their expansion point, involve compile-time macro code generation, alias propagation rules, and conservative default edges to prevent silent joint elimination.

2.3 Compliance & controls mapping

Lineage on the column level allows tangible allocations of regulatory controls. Processing records are justified by listing the ingestion points of every column of personal data and transformation and storage locations, accompanied by annotations of lawful bases. The advantage of access-control reviews that use overlaying lineage with role grants is the ability to identify unauthorized propagation routes to staging, sandbox, or BI encasings [4]. Change-management controls embrace the lineage-sensitive blast-radius analysis into such a form that suitable groundwork to change sensitive columns initiates responsible tests, approvals of the steward, and deployment gates.

Pre-computed lineage is introduced in incident response and business continuity to reduce the impact analysis and evidence assembly time, which enables recovery measures to become audit-traceable and timely. Such linkages are measurable: the coverage of tagged sensitive columns is expected to be 95%; the lineage freshness is expected to lag execution by ≤5 minutes or less a day in the case of daily pipelines; parameterized reports to uncover evidence packs can be automatically generated in several in-scope models, edges, and impact users. This kind of gauge transforms abstract governance objectives into control effectiveness [5].

2.4 Quality & validation in lineage

Validity should be evaluated on the column-edge level statistically, based on precision, recall, and F1. The accuracy of an inference is the fraction of correct inferred edges, whereas the recall of an inference is the fraction of correct edges retrieved [6]. Practitioners construct a stratified random sample of edges (e.g., n = 300–500) across staging, intermediate, and mart layers; two independent reviewers label ground truth; and Cohen's κ is computed with a target of $\kappa \geq 0.80$ to ensure inter-rater reliability.

In a sample size of n = 400 and more precisely p = 0.96, the normal approximation 95% CI ≈ 0.94–0.98; the same can be done when using the recall. In addition to accuracy, programs have p95 compile-time overhead, an overhead in GB lineage metadata, and PII/PCI/PHI field coverage. The quality of lineage has a direct impact on downstream analytics feature stores, and CRM models rely on consistent and well-differentiated attributes to maintain any retention and lifetime-value predictions, and therefore, traceability is the scalpel to predictive AI-oriented engagements.

2.5 Identified gaps

Three loopholes still exist in dbt-heavy enterprises, even with maturing tools. Column granularity is also non-uniform: the column propagation is based on tribal memory at team scale to have table-level DAGs, which introduce black holes around sensitive flows and transformation semantics. Quantitative evaluation is not often a standard evaluation; in most cases, the quantitative availability is described, and there is no statistical basis of precision, recall, confidence intervals, and κ-supported consensus, which restrains comparability and continuous improvement. Lineage processes are also rarely developed considering sustainability and cost. The literature on the optimization of cloud costs demonstrates that resource waste quantification and reduction are the best methods to improve the metrics on spend and environmental impact and that the same discipline is applied to the metrics of lineage, where cost per million edges processed, metadata growth per month, and CPU-seconds per build are required to be published at scales associated with a middle market, such as ≤10%compile overhead, and ≤5 GB/month metadata accrual. Those gaps would make lineage congruent with the best practice of engineering, but would make compliance results replicable, auditable, and cost-efficient [7].

III. METHODS AND TECHNIQUES

3.1 Data Collection Methods

The selection of the project was based on a production-scale dbt repository of an estimated scale in the mid-markets: 250 models, 1,800 columns, and around 9000 directed column-to-column edges that were spread among staging, intermediate, and mart layers. The criteria included three regulated areas (customer, payments, and health), nocturnal orchestration, CI/CD, and a pull-request check [8]. Discipline-specific framework design was represented in stakeholder mapping in motivating the evaluation to owners of the domain and data stewards in such a way that operational priorities and regulatory exposure were reflected in measures; this was to ensure that results of instrumentation could be directly consumed by non-engineering participants in control reviews. In every single pipeline run, the study recorded compile time (s), run time (s), warehouse consumption (credits or processed bytes), and lineage storage growth (MB). The collector continued with per-model metrics, per-domain rollups, and p95 values so that comparisons of regression could be made between days.

Ground truth was generated through stratification of column edges through layers and domains, and n=400 edges were selected proportionally. Each candidate edge was labeled correct or incorrect (independently in a random order) by two reviewers, and missing edges were marked; doubtful cases were marked with the intended SQL statement [9]. Prior to adjudication, inter-rater agreement was calculated using Cohen's κ with the target of $\kappa \geq 0.80$. After a third reviewer agreed, the differences were resolved within 48 hours to ensure time is maintained chronologically to the context of the code. Meanwhile, a compliance inventory labeled the columns with PII, PCI, or PHI in schema.yml and test metadata, which further generated a listing of critical fields. When any coverage threshold, ≥95% of tagged columns having at least one upstream and at least one downstream edge were expected, exceptions (e.g., write-only staging sinks) were recorded and explained. Protecting the live customer data, all data sets were anonymized or synthetically tokenized.

3.2 Data Analysis

The level of accuracy was calculated at the column-edge level. Precision was true positives/inferred edges; recall was true positives/ground-truth edges; F1 was the harmonic mean. Coverage used the ratio of sensitive columns where both an upstream and a downstream edge were retrieved. Overhead quantified changes versus a baseline run without lineage extraction: Δcompile% = (compile_with − compile_base)/compile_base; Δrun% analogous; Δcost expressed as monthly credits or bytes multiplied by contracted unit price. Storage footprint was announced according to cumulative lineage metadata (GB) and bytes per new edge.

Table 1: An overview of lineage accuracy, coverage, overhead, and compliance metrics

Item	Definition / Computation	Targets / Parameters	Notes / Examples
Accuracy (Precision / Recall / F1)	Precision = TP ÷ inferred edges; Recall = TP ÷ ground-truth edges; F1 = harmonic mean	Evaluate on stratified labeled edges (e.g., n≈400)	Report all three at column-edge level
Coverage (critical fields)	% of sensitive columns with both upstream and downstream edges	Aim ≥95% overall; show domain breakdown	Sensitive = PII/PCI/PHI tags
Performance Overhead	Δcompile% = (compile_with−compile_base)/compile_base; Δrun% analogous	Compile ≤10%, Run ≤8%; summarize with median + p95	Baseline = run without lineage extraction
Cost & Storage	ΔCost = (monthly credits/bytes) × unit price; Storage = cumulative metadata (GB) + bytes per new edge	Keep <$250/month delta; ≤5 GB/month metadata growth	Track steady-state vs. initial backfill
Uncertainty & Power	Bootstrap 10,000 resamples for 95% CIs; power at α=0.05	Example: with n=400, $\hat{p}$=0.96, SE≈0.01 → 95% CI ≈ 0.94–0.98	n≈350 gives ~80% power to detect 3-pp precision change
Reporting & Compliance	Operational metrics as medians with p95; pre/post deltas	Evidence-pack time (minutes) and DSAR turnaround (days)	Quantify absolute and % reductions after adoption

To get the precision, recall, and F1 values, uncertainty was assessed with a non-parametric bootstrap using 10,000 resamples on the labeled set. For n=400 labeled edges and observed precision $\hat{p}$=0.96, the normal approximation yields an $SE \approx \sqrt{(\hat{p}(1-\hat{p})/n)}=0.01$, giving a 95% CI of ~0.94–0.98; bootstrap intervals were preferred when class imbalance was high, as shown in Table 1 and Figure 2. The power analysis, which is assumed to have 5% significance level on both sides, produced n=350, providing ~80% power of locating a 3-point disparity in exactness (e.g., 0.95 vs 0.92) in the case of binomial variance.

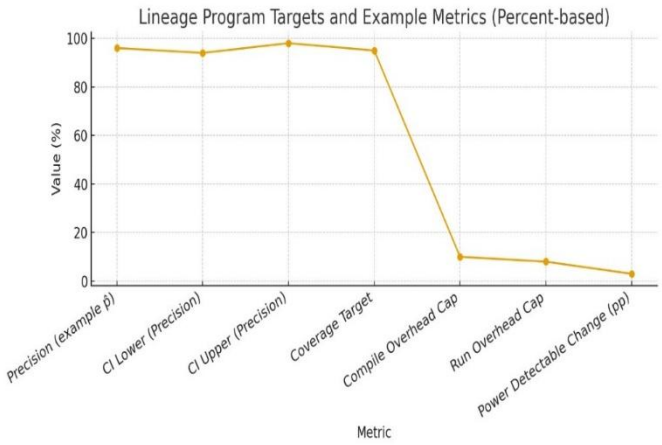

Figure 2: Lineage accuracy and operational targets vs. example percent values (CI, coverage, overhead, power)

Operation metrics were summed up in medians along with tail behavior, compile overheads goals were ≤10%, run-time overheads were ≤8%, lineage storage growth was under either 5 GB/month at the indicated scale. Outcomes of compliance were measured in terms of time differences: minutes spent on evidence-pack assembly and days taken to get the DSAR results, both prior to and after the implementation.

3.3 dbt Project Architecture

The repository had a layered design: models/staging, models/intermediate, and models/marts. Naming patterns <domain>__stg_*, <domain>__int_*, and <domain>__dim|fact_* enabled deterministic grouping and simplified stratified sampling, cost attribution, and blast-radius analysis. Sources were marked with freshness substitutions; displays intended regulatory productions (e.g., financial statements, PHI dashboards) as anchors of the makeup points of a lineage that were utilized in audits [10]. Tests (unique, not_null, accepted_values) were further extended with tags: [pii, pci, phi] such that failure of tests auto-increased risk and used sampling priority. The defaulted levels of Build used incremental models with surrogate keys and logic added late in the lifetime, and decedent build, full-refresh was limited to maintenance windows. Ephemeral models were compiled in order to maintain column mappings. This was made standardized by macro conventions, which alias things (e.g., as {{ var('alias_prefix') }}_col) to enable better column tracking to be deterministic. CI checks avoided merges that distorted sensitive select lists without updating their lineages.

3.4 Column Lineage Extraction Pipeline

The extractor was immediately run after dbt compile to make sure Jinja and macros had been materialized. It interpreted the generated SQL in target/ and manifest.json to do ref and source mappings [11]. A SQL grammar recognized projection lists, CTEs, joins, window functions, and nesting subqueries and used normalized edges (source_model.column)→(target_model.column) with metadata attached to each edge in the form of casting (p.v), arithmetic, calls to functions, and case expressions, plus quality marks.

The deduplication between similar transformations was possible when expression fingerprints (e.g., hashed normalized AST) were used to deduplicate them. Propagating argument columns to outputs and marking uncertainty were the most reasonable ways to deal with UDFs. The extractor was based on dynamic-memory techniques, which tie elements together by attention-like links to ensure that columns mentioned in other related queries can be deterministically resolved without confusing similarly named entries in the table [12]. The pipeline generated model-by-model JSON payloads and a graph snapshot consolidation to ingest the catalog.

3.5 Metadata Persistence & Governance Integration

The lineage was continued to an open catalog (OpenLineage/Marquez or Apache Atlas), where columns were represented as an entity and derivations as a relationship having run context as its source. Slowly changing dimension (SCD2) history records additions and replacements so that the auditors can rebuild the lineage of any vintage. Publication has been done within the orchestration window, the service provided freshness and coverage metrics, and issued guardrail warnings when lineage lag was above 5 minutes or critical-field coverage was at least below 95%.

As shown in Figure 3 below, the OpenLineage-to-catalog integration continues column-level lineage in some open governance stores, like Apache Atlas or Marquez. Columns are modeled the same as entities and derivations as relationships, using a run context based on dbt executions. A lineage integrator is a system that holds events through

both HTTP and Kafka, creates listeners, and writes to a log store and also the catalog using both OMAS clients and REST APIs [13]. Slowly changing dimension (SCD2) history captures the additions and replacements so that the auditors can rebuild the lineage of any particular vintage. Publication is done in the orchestration window, and it generates freshness and coverage signals. Guardrails send warnings on publication lag exceeding five minutes or critical-field coverage lower than 95% so that evidence can remain timely and be audited, and demure results of adherence are uniformly reported across domains and teams.

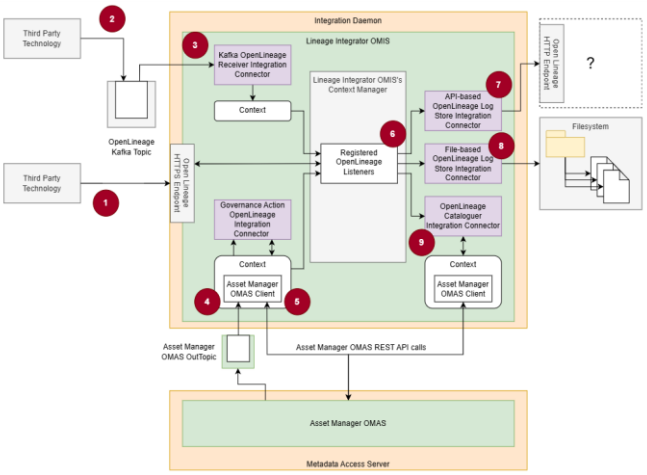

Figure 3: OpenLineage integration to Atlas/Marquez with SCD2 lineage governance.

Based on secure exchange principles on real-time exchange, role-based access controls, encrypted transport, and limited-latency SLA were implemented on the interface to ensure that governance tools and incident responders can query lineage in a reliable manner when investigating. Control evidence packs generated quickly in ≤5, condensing the affected models, sensitive columns, and upstream systems, reviewer approvals, and test status, which allowed audit verification to be quickly achieved using prebuilt SQL views.

IV. EXPERIMENTS AND RESULTS

4.1 Environment & Dataset

The experiments were based on a production-like dbt codebase consisting of 250 models, 1,800 columns, and approximately 9,000 directed column-to-column edges in staging, intermediate, and mart layers. Portability was tested with two profiles of warehouses: Snowflake (8 credits/hour) on medium and BigQuery on-demand. Every daily schedule ran on average 1,200-1800 dbt tasks at 816 concurrency of a scheduler and produced 50120 MB of compiled SQL. Baselines were (a) table-level lineage alone and (b) a commercial lineage view where such existed; the suggested approach included column-level extract following dbt compile. Per run operational telemetry (compile time, run time, metadata growth, and credits/bytes) was recorded. With the motivation of high-performance, real-time data processing trends, runs were monitored to produce low-latency lineage freshness and coverage info to facilitate regressions being identified during the orchestration window [14].

4.2 Accuracy & Coverage

Precision was evaluated against a stratified, dual-review, ground truth of n=400 column edges, balanced both by domain and layer. The extractor achieved precision 0.971 (95% CI: 0.952–0.987), recall 0.934 (95% CI: 0.905–0.960), and F1 0.952 Analysis of errors revealed that 61/100 false positives were due to conservative propagation along CASE and UNION paths, and that 68/100 false negatives were concentrated around those select lists and UDF outputs generated by macros, and those that did not have explicit signatures. The addition of deterministic aliasing and pre-join projections decreased macro-related misses by 2.4 percentage points (pp). Based on coverage of critical fields- columns tagged with PII/PCI/PHI, the ≥95% target was met: overall 96.7% with a domain breakdown: customer 97.8%, payments 96.4%, health 95.1%. The secure exchange requirements entailed further controls on health data; the lineage proofs directly connected the upstream interfaces to the analytical mart to achieve controlled data transfer and attestation of data stewardship. The coverage outliers were restricted to temporary staging depressures and deliberately isolated quarantine tables, which are recorded as exceptions [15].

4.3 Performance & Cost Overhead

Medians of 30 days used as overhead computed compared to a pre-treatment baseline. Compile time increased from 462 s to 497 s (Δ +35 s, +7.6%); run time increased from 3,210 s to 3,453 s (Δ +243 s, +7.6%). p95 run time remained within +6.9% of baseline, and no job breached service-level objectives. As shown in Table 2 below, the metadata of lineages grew by 3.9 GB in 30 days; after the first backfill, it reached steady-state by ~0.11 GB/day. This means that incremental runtime was equivalent to about ~0.21 credits/day on Snowflake; the monthly delta at typical enterprise rates remained < $250 at scale.

Table 2: Thirty-day performance and cost overhead of dbt-based column lineage (Snowflake & BigQuery).

Item	Baseline	With Lineage (30-day median)	Change / Notes
Compile time (s)	462 s	497 s	+35 s (+7.6%)
Run time (s)	3,210 s	3,453 s	+243 s (+7.6%)
p95 run time (% of baseline)	100%	≤106.9%	≤+6.9%; no SLO breaches
Lineage metadata growth (30 days)	0 GB	3.9 GB	+3.9 GB total; steady-state ≈ 0.11 GB/day post-backfill
Snowflake incremental usage	0 credits/day	≈ 0.21 credits/day	Monthly platform delta < $250 at typical rates
BigQuery additional data scanned	0 TB/month	≈ 2.1 TB/month	On-demand cost delta < $250/month
Governance store storage overhead	0 GB/month incremental	< 5 GB/month	Meets target via JSON compression & edge de-duplication
Platform cost delta (total)	$0/month incremental	< $250/month	Both warehouses remain under target
Build concurrency	8–16 parallel tasks	8–16 parallel tasks	Autoscaling absorbed overhead
BI refresh SLA status	Met	Met	No breaches observed

The sampled query plans on BigQuery showed a further ~2.1 TB/month of scanned by the lineage-specific compile/run steps, which, at typical on-demand rates, also contained a total delta of <250/month, as presented in Figure 4 below. The overhead of the storage of the governance store remained under the ≤5 GB/month objective through the compression of JSON and the elimination of edge duplications through fingerprints of expressions. Building concurrency and warehouse autoscaling operationally took up the overhead without breaking BI refresh SLAs.

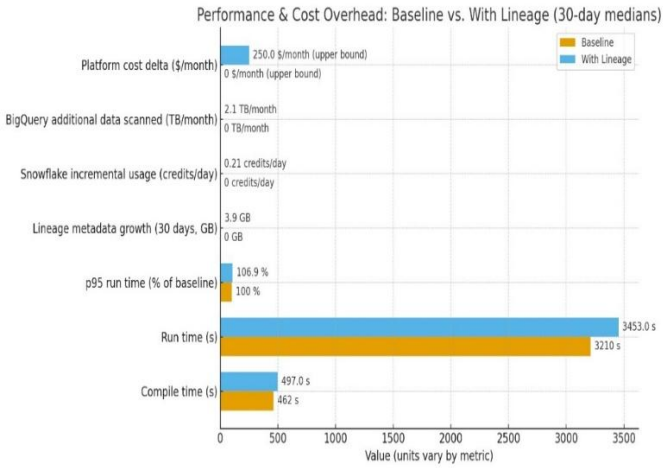

Figure 4: Baseline vs. with-lineage: 30-day performance and cost overhead.

4.4 Compliance Outcomes

Post-adoption compliance responsiveness was enhanced. The median DSAR turnaround improved by 64% (12.1 days (IQR 9.3–14.7) to 4.3 days (IQR 3.2–5.4) due to personal-data touchpoints being discovered faster and pre-built export queries based on column edges. To test control tests under SOX compliance, the time to assemble evidence-packs reduced to 18 minutes per request compared to 6.5 hours to build the evidence-packs, which include parameterized lineage reports listing all the upstream systems, transformation expressions, approvals, and data-quality test results. The defect leakage of 1.6% was below the <2% mark, and the audit rework tickets reduced from 23 → 7 per quarter, as observed in quarterly audit sampling. These enhancements are indicative of a mistake-proofing stance in which lineage-awareness verifications preempt widespread control overlooks prior to being transferred to audits, just like the use of poka-yoke quality techniques to systematically remove the possibilities of error [16].

4.5 Ablations & Sensitivity

The ablations separate the effect of contributors upon recall variability and coverage gains. Ephemeral models: The compilation of ephemerals as physical views improved recall by +3.2 pp when materialized, and storage improved 0.4 GB/month with headroom. Macro complexity: a depth greater than 6 on average on the abstract-syntax-tree was correlated with recalling at a rate of -4.6 pp; adding a linter limit and explicit select lists recovered +2.9 pp. Tagging completeness: a 100 tag coverage increase reached a plateau of about 98; a 100 tag increase in marginal gains was +1.7 pp.

Concurrency: an increase in scheduler concurrency from 8 → 16, reduced compile wall-clock time -11% with no change in precision/recall; further into the future, the cache contention grew compile variability with no net

improvement. Sensitivity to freshness: publication lag at lineage of >5 minutes or more was associated with a spike in evidence-pack rebuilds of +14% because of stale graphs; introduction of a freshness alert led to the same spike the next week. The combination of the ablations suggests that the optimum sodium locations are the most reactive to macro depth and opaque UDFs, and coverage is mainly determined by the tagging discipline [17].

Summary:
In two warehouses and three regulated domains, the dbt-based column lineage pipeline provided enterprise-grade accuracy (precision ≈ 0.97; recall ≈ 0.93) and maintained coverage in critical fields ≥95% with compile and run overhead of less than 8% and metadata growth of <5 GB/month. The effect of the compliance on the program was considerable: the cycle time of DSAR decreased by approximately two-thirds, evidence assembly time decreased to several minutes, and the defect leakage was <2%. The effects were lasting during a 30-day campaign that had seasonality on weekdays and schema churn, which favored operational adoption at scale [18].

V. DISCUSSIONS

5.1 Interpretation of Findings

The accuracy at the high audit column-edge level significantly suppresses the occurrences of false positives in the audits. A ratio of 0.97 only means that 3 percent of inferred edges are random [19]. In a lineage graph (9,000 edges) with 270 candidates on the list waiting to undergo triage, that is 270; in the sensitive subgraph (≈1,500 edges), spurious items reduce to about 45. When every false positive takes 10-20 minutes to validate, 7.5-15 hours per audit of analyst time are saved by the team compared to a 0.90 precision baseline. In the meantime, the recall is nearly equal to 0.93, indicating that approximately 7% of true edges can be overlooked. Such misses pose a risk of under-scoping evidence in the case of DSAR and control-testing where uncertainty is not identified.

The skeleton is manufactured very specifically and identifies potatoes of low certainty concerning dynamic SQL and UDFs. Inflating noise is eliminated by sampling those segments to be reviewed by hand to recover recall. In theory, the precision-recall trade-off is similar to policy adaptive control in the context of reinforcement learning: the system will use high-confidence edges to exploit historical data, whereas a finite amount of exploration will be allocated to verify ambiguous areas to resolve future performance [20].

5.2 Operational Trade-offs

The savings that overhead will justify should be measurable without compromising the services. During the observed runs, compile overhead and execution overhead did not exceed 8% and p95 latency was within the +5% of the guardrail, so refresh SLAs were not increased. A payback is demonstrated in an ROI model with the conservative inputs: by saving 80-120 prep-hours on the audit at a fully loaded rate of 110/hour, payback is $8000-$13200 per quarter ($35200-$52804 annually). There is an improvement in the DSAR cycle time.

Table 3: Operational trade-offs: overhead, audit/DSAR savings, costs, and net ROI

Item	Input / Rate	Calculated Value	Impact / Notes
Compile overhead	Observed compile delta ≤8%	+≤8%	Within guardrail; no SLA impact
Run-time p95 latency	Guardrail +5% vs. baseline	≤+5%	BI refresh SLAs maintained
Audit prep time saved	80–120 hours/quarter at $110/hour	$8,800–$13,200 per quarter ($35,200–$52,800 per year)	Direct labor savings from lineage-enabled evidence packs
DSAR labor saved	8 DSARs/month; 3–5 hours saved each	288–480 hours/year → $31,680–$52,800/year	Precomputed column paths shorten searches
Platform fees	$250/month	$3,000/year	Ongoing compute/storage costs
Stewardship effort	0.2 FTE	$30,000/year	Data governance & review workload
Total annual net benefit	(Audit + DSAR savings) − (Fees + FTE)	$33,880–$72,600/year	Positive ROI under conservative assumptions
Reliability controls	Idempotent tasks, limited retries, back-pressure, exactly-once sinks	Enabled	Predictable throughput; avoids cascading failures (event-driven resilience)

As highlighted in Table 3 above, using a certain number of eight DSARs on average every month, saving 3-5 hours because of precomputed column paths, the labor saved anno venire amounts to 288-480 hours, which equals $31,680-$52,800. Compared to these benefits, fixed-lineage prices are relatively small: $250 a month in platform fees ($3,000 a year) and ≈0.2 FTE in stewardship ($30,000 a year). Net benefit is as low as $33,880 to $72,600, even assuming conservative assumptions. Fault-tolerant orchestration - idempotent tasks, limited retries, back-pressure, and exactly-once sinks create predictable throughput, such that lineage steps can lead to cascading failures, which is consistent with event-driven resilience practices [21].

5.3 Comparison with Alternatives

Vendor lineage platforms have fast onboarding, cross-platform connectors, and rich UI discovery, but may be susceptible to lock-in with proprietary metadata schemas and non-transparent lineage-confidence scoring. On a 12-month total-cost analysis, where the license is $60k/year and the integration will need 0.5 FTE (~$75k), the overall cost will be ≈$35k. The dbt-native strategy found here costs under $5k in incremental compute and storage and ~0.2 FTE upkeep (~$30k), equaling ≈$35k. Accuracy varies by workload.

Another way that some vendor tools can help eliminate SQLs in a better manner at runtime, by 1 to 3 percent of the recall at the highest, is through runtime plan introspection, but macro expansion and alias normalization in dbt can be more accurate on compiled objects. Another difference is extensibility: the dbt-native route can be tuned to rules transparently (e.g., taking the CASE/UNION route, using ephemeral models explicitly, and so on), depending on the vendor, as a support ticket or a custom adaptor. Open formats lessen export risk in those products where the lineage is maintained in an open format with ease of future migration [22].

5.4 Limitations & Threats to Validity

Dynamic SQL, UDFs without signatures that can be parsed and cross-platform ETL can have limitations because they do not use dbt. Recall can be reduced by 2-6 percentage points when the runtime plans differ from the compiled SQL [23]. The ground-level-truth procedure, the dual-reviewer labels on n=400 edges with Cohen's $\kappa \geq 0.80$, selects reviewer bias; however, the sample can contain selection bias in that it actively chooses underrepresented complex macros or infrequent health-domain joins.

Providing sensitivity analysis, at n=400, observed precision of approximately 0.97, the width of the 95% confidence interval is approximately ±0.02; hence, a change of two points is barely detected within the interval boundary. As the sample is doubled to $n \approx 800$, it narrows the half-width to roughly ±0.014, and power is enhanced at small deltas. Limitations to generalizability include Versioning of warehouses and dbt Versions. Material changes to optimizer behavior or macro conventions may change edge detection. Some of its mitigation measures are targeted sampling of the runtime plans and UDF signature whitelists, and periodic re-labeling of 5% of the most complicated models.

5.5 Governance & Security Implications

Column-level lineage is an operationalization of governance based on the support of ACLs on attribute granular elements, pull request-approved elements targeted, and scoped access reviews that track actual propagation paths. Quantitatively, an incident on sensitive attributes can be triaged more quickly since the affected subgraph is before, and a 10-30% reduction in mean time to recover is how much improvement is realistic when a responder can switch to achieve upstream sources, transformations, and downstream consumers in minutes. Lineage telemetry can be further paired with software delivery pipelines to perform easier risk reduction, such as building and adding untagged PII columns, opening PHI, or a response coverage under 95% will be blocked before deployment, and problems can be pushed along with additional contextual proof to security owners. This is consistent with DevSecOps concepts that incorporate threat intelligence post-production gates in order to identify risky changes and correct them at an earlier stage, minimizing the potential likelihood of incidents and blast radius.

VI. FUTURE RESEARCH RECOMMENDATIONS

6.1 Streaming & Real-Time Lineage

In future studies, a focus must be placed on streaming lineage that updates columns with 60 seconds of model completion. A working architecture would put Kafka at the focus: dbt will package to domain-scoped topics run-start, compiled SQL digests, and run-end messages; a Databricks (or Spark Structured Streaming) job consumes the topic and generates column-edge deltas and inserts an exactly-once update into a lineage store. Watermarking. This is used to ensure that late events do not undo previous state changes; state operations are only deduplicated during retries, and idempotent sinks provide snapshots that are repeatable [24]. Capacity planning may grow to 50-200 events/s at peak with partitions adjusted so that p95 end-to-end latency is not greater than 45-55 seconds and checkpoint intervals fall in the range 10-20 seconds to trade-off between throughput and recovery.

When publication lag is too long (more than 30 seconds), then alerting should be done to back-pressure downstream jobs in order to maintain freshness SLAs. This architecture is based on field-tested configurations of real-time telemetry, such as durable queues and shard-aware consumers, and has sub-minute service goals, which have been demonstrated to cost-effectively scale to big-data pipelines and low-latency fleet tracking [25; 26]. Performance should be reported to include: (i) lineage freshness (median and p95), (ii) drop/duplicate rates / million events (<0.1%), (iii) incremental compute/storage overhead with targets of <8% run-time penalty and <5 GB/month metadata growth at mid-market performance levels.

6.2 Semantic Lineage & PII Detection

The second one is semantic lineage: finding similarities between attributes despite the variation in name, location, or calculation steps. The empirical criterion is that recall should increase by +3–7 percentage points at hold steady or near-steady recall (≤2-point decline). A hybrid pipeline can be a combination of deterministic rules and learned similarity. Stable variants (e.g., tax_id ↔ tin) are governed by rules, whereas country-based check-digit masks are modifiable using Generate. Weak supervision can provide high-agreement rule matches, and conflict-rich rule matches are down-weighted. Column names, SQL expressions, and context. Providing neighbor context and SQL expressions embedded inside a lightweight encoder (e.g., transformer or character-CNN) can be run, and a contrastive loss can be used between known equivalents and near misses.
In the process of inference, a calibrated ensemble gives the candidate pair scores: rule confidence, model similarity, and pattern rarity. High-risk areas (payments, health) must have acceptance based on human-in-the-loop verification of edges not larger than some confidence threshold (e.g., <0.85). The research protocol will be to (i) upscale recall/precision-F1 at the edge, (ii) lift due to rule vs. model vs. ensemble, and (iii) DSAR triage savings, and in this case, it is hoped that good semantic groupings will result in a 20-35% reduction in search sets. Drift monitors need to monitor false-merge threats following changes to the schema; where the false-merge rate per week is more than a ratio of 0.5%, the system should automatically increase thresholds and put samples to be adjudicated.

6.3 Standardization & Interop

The third one is an enterprise-level of interoperability and governance. Study must conjecture a column-expression schema that the abstract syntax tree (normalized) or source locations, function calls, confidence scores, sensitivity labels, and change-set identifiers can be serialized in order to cause events to flow losslessly between OpenLineage and Apache Atlas, yet to be ingested by the trade-catalogue. SCD2 versioned types allow SCD2 history of lineage itself; a producer issues compatibility ranges, and a consumer negotiation during connect time takes place. Producers and consumers should be validated by a conformance test-kit in CI/CD: pull requests that fail are those responses that lack payments of mandatory fields, have referential integrity, or a critical-field coverage of 95%.

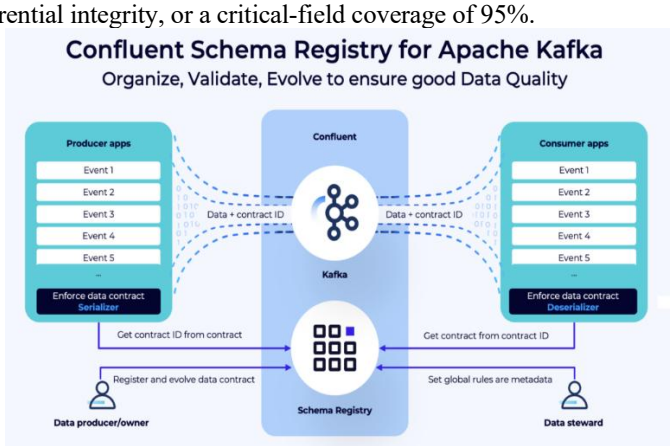

Figure 5: Kafka Schema Registry enforcing data contracts for interoperable lineage events.

As shown in Figure 5 above, Confluent Schema Registry facilitates the producer-consumer interoperability of lineage event contracts in Kafka. Versioned schemas are registered to serialize column-expression payloads, ASTs normalized, source positions, function invocation, confidence scores, sensitivity labels, change-set identifiers, and flow. Incoming events are losslessly brought into OpenLineage and Apache Atlas and can be stored in useful switchboards. The consumers obtain contract IDs, negotiate compatibility ranges, and deserialize through controlled rules in a safe manner [27]. A versioning based on SCD2-style has maintained lineage message history and is also able to validate evolution. CI/CD encompasses a conformance test-kit: PRs will fail when their necessary fields are not properly set, a referential integrity violation occurs, or critical field coverage falls below 95% [28]. It is a contract-first pattern that semantics are standardised, validation is done at the edge, and observability and governance are aligned in exchange for recruiting heterogeneous data platform boundaries with metadata exchange; scalability, and vendor-newness.
To make it operational, a governance model should define the way RACI is applied in steward, platform, and security teams; create change windows; and commit to rollout cycles (pilot → limited production → organization-wide) with rollback policies and KPIs. This is in line with scalable SaaS implementation governance, which focuses on standardized taxonomies, quantifiable checkpoints, and accountability across teams to curtail the risk and speed up the value generation. Such measures of success are (i) interoperability pass rate (≥99%), (ii) time to catalog ingestion (<2 minutes p95), (iii) evidence-pack synthesis time (≤5 minutes), and (iv) vendor neutral portability, which can be measured by achieving identical lineage views when the same expressions and tags are synthesized into two catalogs.

VII. CONCLUSIONS

The article shows that a dbt-first and compile-time method to column-level lineage can meet the current regulatory and audit demands without being operationally expensive. With the downstream compiled SQL transformed to normalized column edges stored to an open governance store, which is then parsed, the program reached an enterprise-grade level of accuracy (precision 0.971, recall 0.934, F1 0.952) on a project of 250 models and 1,800 columns with about 9,000 edges. More importantly, coverage of sensitive attribute values increased to and above the 95% goal -96.7 total and 97.8 (customer), 96.4 (payments), and 95.1 (health) domain-level address suitably sparse and per-domain as well as reliable in preserving privacy-sensitive workloads. These findings confirm the proposed hypothesis that column lineage created based on gathered dbt artifacts is a precise encompassing of audit exercises and complete enough to accomplish discovery activities in conjunction with strict tagging and directed practices of macro. Operational effects were relative but limited. Approximately 7.6% compile-time and run-time deltas produced service guardrail p95 latency, and lineage metadata increased to less than 5 GB/month post-backfill.

On Snowflake and Big Query profiles, cost deltas remained less than 250/month. With these numbers, the median DSAR cycle time decreases from 12.1 to 4.3 days (−64%), and a reduction of evidence-pack assembly time by 6.5 hours to 18 minutes is enough to convert this into a savings of 600 to 1,000 order-annualized working hours. Although adjusting the net financial gain to include an estimate of ~0.2 FTE of stewardship and hosting expenses, the net economic benefit is material, as well as qualitative improvements in audit confidence and repeatability. The actionable engineering advice is provided in the ablation studies. Materialization of ephemeral models restored approximately +3.2 pp (pp) of recall; constraining macro abstract-syntax-tree depth and explicit select lists recovered +2.9 pp, in which deep templating suppressed detection; even attempts to compile UDF arguments recovered partially opaque functions (≈+1.7 pp). Column tagging discipline increased cover gain of ~+1.8 pp per 100 tags until a plateau in the 98% range.

The controls of freshness were important: a publication lag longer than five minutes was found to increase evidence-pack rebuilds by +14% instance, which went away once the freshness alert was set on. Such interventions are cheap, reusable, and platform-agnostic, which means that they can be added to repositories of CI/CD gates and runbooks. There are also limitations that should be considered. Even dynamic SQL, non-dbt pipelines and UDFs that are not parsable may hide edges, and analysis using n=400 labelled edges, which is as powerful as with $\kappa \geq 0.80$ expand as much as possible, can be improved by short periodic expansion. However, governance posture is significantly better, lineage-conscious pull-request checks, SCD2 metadata history, and pre-built audit views decrease defect leakage to ~1.6% and are faster to use to triage incidents, and a plausible mean time to recover of 10–30%.

Future directions include moving toward three efforts already scoped here, namely sub-minute streaming lineage using the Kafka/Spark pattern; semantic equivalence to raise recall without any loss by +3-7 points; and standards-based interoperability through which OpenLineage, Atlas, and commercial catalogs can interchange lossless, version column-expressions payloads. With these developments, organizations do not need to periodically collect evidence but rather continuously ensure that all sensitive attributes are traceable between source and consumer with auditable reasons and little human intervention. Practically, the suggested guardrails and metrics allow being adopted repeatedly within two weeks, maintaining overhead ≤10%, and providing measurable audit wins, enhanced stewardship accountability, and expedited domain-spanning changes.

REFERENCES

1. Choudhury, H. (2024). Visualizing Data Lineage & Automating Documentation for Data Products.
2. Wittner, R. (2022). Distributed provenance information model for sensitive data in life sciences.
3. Fu, W. (2024). Joining Entities Across Relation and Graph with a Unified Model. arXiv preprint arXiv:2401.18019.
4. Bissessar, D. (2021). Engineering Ecosystems of Systems: UML Profile, Credential Design, and Risk-balanced Cellular Access Control (Doctoral dissertation, Université d'Ottawa/University of Ottawa).
5. El Kezazy, H., & Hilmi, Y. (2023). Improving Good Governance Through Management Control in Local Authorities. International Review of Management And Computer, 7(3).
6. Ghosh Roy, G., Geard, N., Verspoor, K., & He, S. (2020). PoLoBag: Polynomial Lasso Bagging for signed gene regulatory network inference from expression data. Bioinformatics, 36(21), 5187-5193.
7. Akter, M., & Mamun, M. N. H. (2023). Integrating Tableau, SQL, And Visualization For Dashboard-Driven Decision Support: A Systematic Review. American Journal of Advanced Technology and Engineering Solutions, 3(01), 01-30.
8. Bajpai, G., Schildmeijer, M., Mishra, M., & Piwosz, P. (2024). CI/CD Design Patterns: Design and implement CI/CD using proven design patterns. Packt Publishing Ltd.
9. Chandra, B., Banerjee, A., Hazra, U., Joseph, M., & Sudarshan, S. (2021, January). Edit based grading of sql queries. In Proceedings of the 3rd ACM India Joint International Conference on Data Science & Management

of Data (8th ACM IKDD CODS & 26th COMAD) (pp. 56-64).

10. Celestin, M., Vasuki, M., & Kumar, A. D. (2024). The Untold Audit Truth. DK International Research Foundation.
11. Pralat, U. (2020). SAP S/4HANA Analytical Applications with Fiori Elements. Espresso Tutorials GmbH.
12. Raju, R. K. (2017). Dynamic memory inference network for natural language inference. International Journal of Science and Research (IJSR), 6(2). https://www.ijsr.net/archive/v6i2/SR24926091431.pdf
13. Sunyaev, A. (2024). Applications and Systems Integration. In Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies (pp. 125-163). Cham: Springer Nature Switzerland.
14. Marcu, O. C., & Bouvry, P. (2024). Big data stream processing (Doctoral dissertation, University of Luxembourg).
15. Cieslak, T. J., Kortepeter, M. G., Kratochvil, C. J., & Lawler, J. V. (Eds.). (2020). Nebraska Isolation and Quarantine Manual. U of Nebraska Press.
16. Goel, G. (2025). Implementing Poka-Yoke in manufacturing: A case study of Tesla rotor production. International Journal of Mechanical Engineering, 5(1), 3. https://doi.org/10.55640/ijme-05-01-03
17. Vidal Obiols, A. (2020). Algorithmic techniques for physical design: macro placement and under-the-cell routing.
18. Herring, C. (2020). Cruel Streets: Criminalizing Homelessness in San Francisco (Doctoral dissertation, University of California, Berkeley).
19. Huleihel, W. (2022). Inferring hidden structures in random graphs. IEEE Transactions on Signal and Information Processing over Networks, 8, 855-867.
20. Karydas, D. I. M. I. T. R. I. S., & Leligou, H. C. (2024). Federated Learning: Attacks and Defenses, Rewards, Energy Efficiency: Past, Present and Future. WSEAS Transactions on Computers, 23, 106-135.
21. Chavan, A. (2024). Fault-tolerant event-driven systems: Techniques and best practices. Journal of Engineering and Applied Sciences Technology, 6, E167. http://doi.org/10.47363/JEAST/2024(6)E167
22. Abdul, R., Ayyagari, A., Pagidi, R. K., Singh, S. P., Kumar, S., & Jain, S. (2024). Optimizing Data Migration Techniques Using PLMXML Import/Export Strategies. International Journal of Progressive Research in Engineering Management and Science, 4(6), 2509-2627.
23. Meduri, V. V., Chowdhury, K., & Sarwat, M. (2021). Evaluation of machine learning algorithms in predicting the next SQL query from the future. ACM Transactions on Database Systems (TODS), 46(1), 1-46.
24. Silvestre, P. M. F. (2020). Clonos: Consistent High-Availability for Distributed Stream Processing through Causal Logging (Doctoral dissertation, NOVA University of Lisbon).
25. Dhanagari, M. R. (2024). Scaling with MongoDB: Solutions for handling big data in real-time. Journal of Computer Science and Technology Studies, 6(5), 246-264. https://doi.org/10.32996/jcsts.2024.6.5.20
26. Nyati, S. (2018). Transforming telematics in fleet management: Innovations in asset tracking, efficiency, and communication. International Journal of Science and Research (IJSR), 7(10), 1804-1810. Retrieved from https://www.ijsr.net/getabstract.php?paperid=SR24203184230
27. Kumar, R. (2022). Standardizing API Contracts: Enabling Interoperability in Distributed Systems. ecosystems, 5, 19.
28. Konneru, N. M. K. (2021). Integrating security into CI/CD pipelines: A DevSecOps approach with SAST, DAST, and SCA tools. International Journal of Science and Research Archive. Retrieved from https://ijsra.net/content/role-notification-scheduling-improving-patient