

RESEARCH ARTICLE

Intelligent ETL Monitoring and Data Quality Validation for Large-Scale Salesforce Integration Projects

Rama Krishna Inmmuri ^{a,*}^a*Sr. Technical Architect , Salesforce, Prosper, Texas 75078, United States of America.***Abstract**

Large Salesforce integration programs require continuous evidence that incoming records remain complete, structurally valid, semantically consistent, and safe for warehouse consumption. This paper develops an intelligent Extract, Transform, and Load (ETL) monitoring framework that combines deterministic data-quality rules with an explainable anomaly-prioritization model. A public CRM sales-opportunity snapshot containing 201 source records is projected onto a Salesforce-compatible multi-object representation. Fourteen naturally missing account values are preserved, while seven controlled fault classes are injected under a fixed seed to stress underrepresented failure modes without representing the data as proprietary Salesforce records. The resulting 100,000-row workload contains 15,955 review-worthy records. The deterministic rule layer recovered all benchmark-defined defects. Account-grouped evaluation on 15,000 records compared logistic regression, random forest, extremely randomized trees, and XGBoost. XGBoost obtained ROC-AUC 0.9995, PR-AUC 0.9976, F2 0.9953, and recall 1.0000; however, expected calibration error was 0.1968, showing that ranking quality did not imply calibrated probabilities. Rule validation, feature construction, and XGBoost inference processed 100,000 records in 0.483 s, corresponding to 207,092 rows/s in the reported environment. SHAP analysis identified account completeness, deal stage, close value, and sales-cycle duration as dominant signals. The design therefore uses machine learning for explainable triage while retaining deterministic rules as the warehouse acceptance authority.

Keywords: Salesforce integration, Extract, Transform, and Load (ETL) monitoring, Data quality, Anomaly detection, Data integration, Explainable machine learning.

Article information:

ISSN: 3107-9466 (Online)

Published by: **Krrish Scientific Publications Pvt. Ltd.**DOI: <https://doi.org/10.71426/jcdt.v1.si1.pp28-35>

Received: 16 Sep. 2025 | Revised: 26 Oct. 2025 | Accepted: 02 Nov. 2025 | Published: 06 Nov. 2025

Copyright ©2025 Author(s).

This is an open-access article distributed under the terms of the [Creative Commons Attribution-NonCommercial 4.0 International License \(CC BY-NC 4.0\)](https://creativecommons.org/licenses/by-nc/4.0/).**1. Introduction**

Customer-relationship-management (CRM) platforms are frequently treated as operational systems of record for sales opportunities, account relationships, products, ownership, and revenue-stage transitions. When these objects are replicated into enterprise data warehouses, data lakes, or downstream business-intelligence platforms, integration correctness becomes inseparable from data quality. A pipeline can complete on schedule and still be analytically unsafe if identifiers are malformed, required dimensions are absent, stage values violate the configured domain, dates are temporally inconsistent, or amounts conflict with business semantics. This distinction is important because data quality is multidimensional: fitness for use depends not only on

numerical accuracy but also on completeness, consistency, timeliness, and interpretability. Wang and Strong formalized this consumer-oriented view and showed that data users evaluate quality through several dimensions rather than a single error rate [1]. Pipino, Lee, and Wang subsequently emphasized the need for operational metrics that combine objective measurements with contextual quality assessment [2].

Large-scale Salesforce integration amplifies these issues. Opportunity data are commonly joined to account, product, user, campaign, and reference objects whose update frequencies differ. Incremental extraction may arrive out of order; late-arriving dimensions can temporarily orphan facts; configuration changes can introduce new stage codes; and mapping defects can silently alter data types or nullability. A quality mechanism placed only after warehouse loading therefore observes failures too late. Systematic data-quality methodologies instead recommend explicit assessment, mea-

^{*}Corresponding authorEmail address: ramakrishnainnamuri@gmail.com (Rama Krishna Inmmuri).

surement, and improvement phases with defined dimensions and remediation responsibilities [3]. Broader treatments of data quality likewise connect profiling, duplicate detection, integrity constraints, data integration, and record linkage as parts of one control problem [4].

The ETL literature establishes the complementary execution perspective. Vassiliadis describes ETL as a workflow of extraction, transformation, and loading activities whose correctness depends on both local operators and workflow-wide properties [5]. Ali and Wrembel later show that optimization, quality objectives, metadata, parallelism, and user-defined transformation logic remain coupled research concerns rather than isolated implementation details [6]. These observations are directly relevant to Salesforce programs: monitoring must not become a serial afterthought that destroys the throughput gains achieved by bulk extraction and parallel loading. Quality checks must instead be measurable operators with bounded latency and explicit relationships to source fields and warehouse acceptance rules.

ETL-specific quality studies further argue that defects should be traced along the transformation path rather than discovered only in reporting layers [7]. This requirement becomes difficult when error types are heterogeneous. Abedjan *et al.* demonstrate that individual cleaning tools often have limited coverage and that orchestrating multiple detectors can materially affect the amount of human verification required [8]. Systems such as NADEEF provide a general platform for expressing heterogeneous cleaning rules and detecting violations [9], while BigDancing addresses the scaling problem by compiling cleaning logic into distributed execution plans and shared operators [10]. Collectively, these studies motivate a monitoring architecture in which deterministic constraints remain explicit but can be complemented by learned prioritization when the incoming stream is too large for uniform manual review.

A practical gap remains between database-oriented data cleaning and modern CRM integration operations. Production Salesforce pipelines often expose row counts, API errors, and job status, yet operational dashboards do not necessarily quantify semantic quality, rank suspicious records, or explain why an anomaly score changed. Conversely, an unconstrained machine-learning detector can produce high numerical performance while learning leakage, source identity, or artifacts of the fault-generation process. The engineering challenge is therefore not simply to “apply AI” to ETL. It is to construct a layered mechanism in which (i) rule violations have deterministic semantics, (ii) anomaly models are evaluated with leakage control, (iii) explanations refer to observable data characteristics, (iv) throughput overhead is measured, and (v) model confidence is not mistaken for data correctness.

This paper develops such a framework for Salesforce-compatible ETL monitoring. The evaluation is intentionally reproducible and does not claim access to proprietary tenant data. A public CRM opportunity snapshot is decomposed into Opportunity, Account, Product, and Agent logical objects. Its naturally missing account values are retained as genuine source-quality defects. Controlled faults are then added only to create sufficient examples of under-represented failure modes: missing product, invalid stage, date-order violations, inconsistent won/lost values, and

malformed opportunity identifiers. A deterministic validator defines the warehouse acceptance boundary, while XGBoost produces a review priority score from raw structural and business attributes. SHAP explanations are used to inspect which characteristics drive that score.

The contributions are fourfold. First, the paper specifies a multi-layer ETL monitoring architecture that unifies object-level ingestion telemetry, deterministic quality gates, anomaly prioritization, and explainability. Second, it formalizes a composite data-quality score, the XGBoost objective, and SHAP-based explanation values and maps each equation to the corresponding pipeline step. Third, it provides a leakage-controlled evaluation in which replicated source records are grouped by original row identity, preventing copies of the same CRM record from appearing across evaluation groups. Fourth, it measures monitoring overhead up to 100,000 records and reports both positive findings and limitations, particularly the gap between XGBoost ranking quality and probability calibration. The resulting design treats learned monitoring as a triage layer, not as a substitute for deterministic data contracts.

2. Literature Review

2.1. Data cleaning as an operational control problem

Modern data cleaning research increasingly treats quality as a system problem involving detection, repair, prioritization, and user effort. ActiveClean couples iterative data cleaning with statistical model training and prioritizes records whose repair is expected to influence the analytical model [11]. This is relevant to ETL monitoring because it rejects the assumption that all dirty records are equally urgent. HoloClean takes a probabilistic approach, combining integrity constraints, statistical signals, and external evidence in a unified repair model [12]. Its contribution is not merely another detector; it illustrates how heterogeneous evidence can be integrated without reducing quality management to one hand-written rule set.

External information can also improve reliability [13], [23]. In enterprise CRM integration, the analogous external sources are master-data services, account registries, product catalogs, and reference dimensions. However, such services may be unavailable during ingestion, so a practical monitor must still operate when only the inbound object and warehouse metadata are present. Raha addresses another operational burden: configuration. It generates and combines detector configurations and uses limited labels to identify erroneous values without requiring exhaustive user tuning [14]. HoloDetect moves further toward learned error detection by using few-shot learning and data augmentation to capture heterogeneous error patterns with limited labeled examples [15]. These systems motivate the learned triage layer in this work, but they also highlight why experimental fault generation must be separated from claims about naturally occurring enterprise defects.

2.2. Anomaly detection and class imbalance

Anomaly detection is broader than data-quality validation. Chandola, Banerjee, and Kumar categorize anomaly methods according to assumptions about normality, labels, contextual information, and collective behavior [16]. Their

taxonomy is useful for ETL because quality incidents may be point anomalies (one malformed key), contextual anomalies (an amount invalid only for a given stage), or collective anomalies (a batch whose missingness suddenly increases). Density-based Local Outlier Factor (LOF) identifies objects whose local density differs from their neighbors [17]; it is powerful for geometric outliers but less direct for categorical business constraints. Isolation Forest instead isolates rare observations through randomized partitions and is computationally attractive for high-volume detection [18]. Both methods can complement rule systems, yet neither naturally explains a stage-specific business rule such as “Won implies positive close value.” This paper therefore keeps explicit constraints for such semantics and uses supervised learning only for prioritization.

Rare-error prevalence also affects model evaluation. In operational ETL streams, most records are expected to pass validation, so accuracy can remain high even when a detector misses important defects. Imbalanced-learning research emphasizes class weighting, resampling, and minority-sensitive metrics [22]. ROC analysis remains useful for ranking performance across thresholds [23], but precision-recall behavior is particularly informative when the anomaly class is relatively uncommon. For this reason, the experiments report ROC-AUC, PR-AUC, F1, F2, recall, specificity, balanced accuracy, Matthews correlation, Brier score, log loss, and expected calibration error instead of a single accuracy value.

2.3. Tree ensembles and explainability for monitoring

Tree ensembles provide a suitable middle ground for tabular CRM monitoring because they model nonlinear thresholds, missingness, and interactions without requiring a large neural architecture. Random forests reduce variance by aggregating decorrelated trees [21]. XGBoost adds regularized gradient boosting, sparsity-aware split finding, and scalable training procedures [19]. In this work, XGBoost is selected as the primary explainable triage model because the user requirement is not merely to flag a record but to identify the structural or business feature that influenced the flag.

TreeSHAP provides an efficient way to obtain additive feature attributions for tree ensembles [20]. The monitoring interpretation is especially useful: a high anomaly score can be decomposed into contributions from account completeness, stage, amount, cycle duration, key structure, and other features. However, explanations do not convert association into a causal diagnosis. A feature can dominate SHAP magnitude because the benchmark encodes a strong syntactic defect, while a subtler real-world semantic error may remain undetected. Accordingly, this paper uses SHAP as an audit mechanism for model behavior and not as a repair policy.

2.4. Gap addressed by this study

The literature provides mature ideas for quality dimensions, rule-based cleaning, probabilistic repair, anomaly detection, scalable processing, and explanation. The remaining integration gap is architectural: Salesforce-oriented ETL operations require these ideas to coexist under explicit acceptance semantics. The proposed design differs from

Table 1: Dataset and evaluation protocol.

Item	Value
Public source opportunities	201
Naturally missing accounts	14
Scaled monitoring workload	100,000 rows
Review-worthy rows	15,955 (15.955%)
Controlled injected faults	9,000
Model-evaluation subset	15,000 rows
Grouped CV	5 folds by source row
Random seed	20260910
Quality rules	7
Compared classifiers	4

a pure data-cleaning system in three respects. First, it preserves a deterministic quality gate whose outcome is auditable independently of a learned model. Second, it measures monitoring latency as part of ETL performance rather than evaluating detection in isolation. Third, it enforces source-row grouping during cross-validation, which is essential when controlled replication is used to construct load-test volumes. These choices target reproducible engineering evidence rather than claims that a single model can discover every form of enterprise data error.

3. Methodology

3.1. Dataset, multi-object representation, and fault protocol

The source snapshot contains 201 CRM sales opportunities with the fields opportunity ID, sales agent, product, account, deal stage, engagement date, close date, and close value. Four logical source objects are derived: Opportunity retains the transaction and foreign-key attributes; Account, Product, and Agent are distinct reference streams. This decomposition represents a Salesforce-compatible multi-object integration pattern but does not imply that the public dataset was exported from a real Salesforce tenant. In addition to business objects, the experiment produces two measured sources of monitoring evidence: validation-event records and execution telemetry.

Fourteen source opportunities have no account value. These omissions are preserved and treated as naturally observed quality defects rather than repaired before evaluation. To obtain sufficient examples of other rule classes, deterministic controlled corruptions are introduced using seed 20260910. Seven injected classes are used: missing account, missing product, invalid stage, close date preceding engagement date, non-positive value for a Won opportunity, positive value for a Lost opportunity, and malformed opportunity ID. The full 100,000-row stream contains 84,045 clean records, 6,955 replications of naturally missing-account records, and 9,000 injected faults. A 15,000-row subset is used for grouped model evaluation; the full stream is reserved for quality and performance measurements. Table 1 summarizes the protocol.

3.2. Deterministic data-quality validation

For each record x , seven Boolean violation functions $v_j(x) \in \{0,1\}$ are evaluated: key format, account completeness, product completeness, stage domain, date order, Won-value semantics, and Lost-value semantics. The batch

quality score is defined as (1), where $Q(B) = 1$ denotes zero rule violations.

$$Q(B) = 1 - \frac{1}{7|B|} \sum_{x \in B} \sum_{j=1}^7 v_j(x) \quad (1)$$

The rule vector is also persisted as an event record containing source-row identity, ingestion identity, rule name, outcome, and execution timestamp. This separation enables the warehouse gate to reject or quarantine failures while retaining evidence for audit and root-cause analysis.

The benchmark label is intentionally aligned with the declared data contract: the 14 naturally missing-account records and all controlled faults are review-worthy. The validator therefore evaluates whether the implementation correctly realizes the contract, not whether the contract captures every possible semantic defect. That limitation is central to the interpretation of the results.

3.3. Explainable XGBoost triage model

The learned monitor does not consume the rule outcomes themselves. Instead, it uses structural and business features derived from the inbound row: deal stage, ID length, number of non-alphanumeric ID characters, account-string length, product-string length, close value, engagement month, engagement weekday, sales-cycle duration, and missing close-date indicator. High-cardinality account names and agent identities are excluded from the classifier because they are not legitimate quality signals and could encourage memorization.

Let $\hat{y}_i = \sum_{k=1}^K f_k(x_i)$ denote the additive tree score. XGBoost minimizes

$$\mathcal{L} = \sum_{i=1}^n \ell(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k), \quad \Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|_2^2, \quad (2)$$

In (2), T is the number of leaves and w the leaf-weight vector.

Class weights are computed inside each training fold. The operating threshold is selected from the out-of-fold predictions to maximize F2, giving recall greater weight than precision because missed data-quality incidents are operationally costly.

Explainability uses additive SHAP values. For feature j , the local contribution is expressed as (3).

$$\phi_j = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S \cup \{j\}}(x) - f_S(x)] \quad (3)$$

Global importance is the mean absolute $|\phi_j|$ over the evaluated records. SHAP is used to verify that the model responds to interpretable quality characteristics rather than arbitrary source identities.

3.4. Leakage control and performance measurement

Controlled volume scaling necessarily creates multiple copies of the same public source record. Random row-wise cross-validation would therefore leak source-specific patterns between folds. To prevent this, each generated row retains its original source-row index and the five-fold split

groups all replicas of the same source row together. The model benchmark uses 15,000 rows because detection accuracy and monitoring throughput answer different questions: the former requires a leakage-controlled labeled sample, while the latter requires stress at larger volume.

Monitoring latency includes deterministic validation, feature construction, and XGBoost inference. For workload size N , throughput is expressed by (4).

$$\Theta(N) = \frac{N}{t_{val}(N) + t_{feat}(N) + t_{inf}(N)}. \quad (4)$$

Three executions are performed per workload size and the median is reported. Hardware-dependent timings are therefore evidence for this reproducibility environment, not a universal Salesforce service-level guarantee.

3.5. Batch-level quality state and alert policy

Row-level quarantine is necessary but insufficient for an operational integration program because a large batch can remain loadable while showing a statistically meaningful degradation in one quality dimension. The framework therefore aggregates every 1,000 monitored rows into a batch state containing the rule-fail rate, $Q(B_t)$, anomaly-score distribution, per-rule counts, and stage-specific missingness. The batch state is stored separately from row-level events so that operators can distinguish isolated defects from a source-wide condition such as a mapping regression or delayed reference object.

A simple alert state can be expressed without allowing the learned model to determine acceptance. Let $e_t = 1 - Q(B_t)$ denote the normalized rule-violation intensity of batch t . A smoothed state is written as (5), where λ controls how rapidly the monitor responds to new degradation.

$$S_t = \lambda e_t + (1 - \lambda) S_{t-1}, \quad 0 < \lambda \leq 1, \quad (5)$$

An operational alert can then combine three conditions: $Q(B_t)$ below a service threshold, S_t above a persistence threshold, or any critical rule whose business impact requires immediate escalation. The present experiment reports the unsmoothed batch measurements and does not optimize λ or an alert threshold against incident labels. Equation (5) is therefore a deployment model rather than a claimed benchmark result.

This distinction matters in Salesforce integration because quality failures are not equally severe. A single missing optional account reference may be quarantined without stopping the pipeline, whereas a sudden appearance of an unrecognized stage code in a large fraction of opportunities can indicate metadata drift and justify pausing downstream transformation. The event schema therefore preserves rule identity and source lineage rather than compressing every defect into one binary anomaly flag. The learned score is useful for ordering investigations inside the set of suspicious rows, but the batch-level state remains traceable to explicit rule counts.

4. Results and discussion

4.1. Quality-rule behavior

The seven declared rules found all 15,955 benchmark-defined review cases with no false positives relative to the

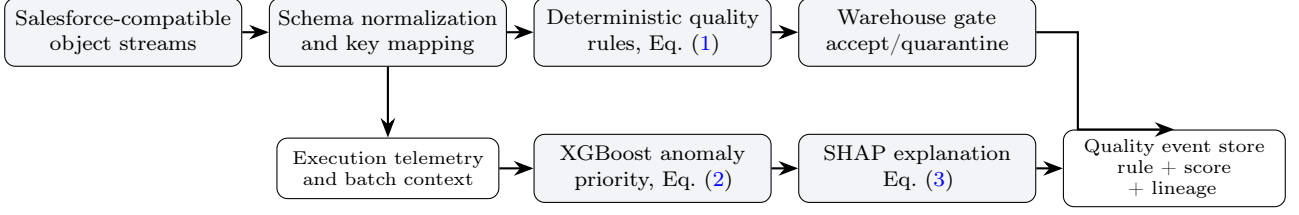


Figure 1: Monitoring flow.

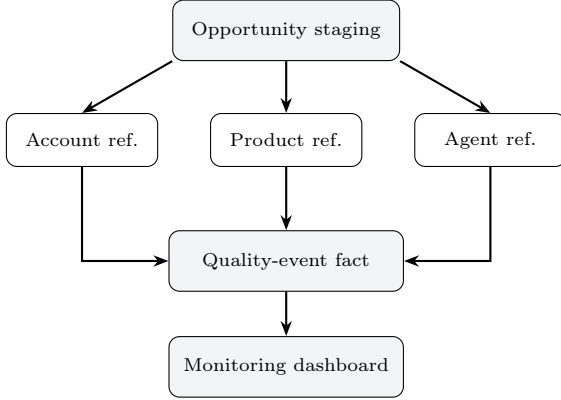


Figure 2: Logical monitoring data model.

Algorithm 1 Quality-aware ETL monitoring

Require: inbound batch B , rule set R , trained model M , threshold τ

```

1 for all  $x \in B$  do
2   normalize schema, types, and business keys
3   evaluate  $v_1(x), \dots, v_7(x)$  using  $R$ 
4   construct non-identity feature vector  $z(x)$ 
5    $p_x \leftarrow M(z(x))$ 
6   compute SHAP explanation  $\phi(x)$  if  $p_x \geq \tau$ 
7   if  $\sum_j v_j(x) > 0$  then
8     quarantine  $x$  and persist rule evidence
9   else
10    admit  $x$  to warehouse loading path
11   end if
12   persist  $p_x$ , lineage, latency, and optional  $\phi(x)$ 
13 end for
14 compute  $Q(B)$  by Eq. (1) and throughput by Eq. (4)

```

benchmark label. This exact result should be interpreted narrowly. The benchmark label was constructed from the known natural missing-account condition plus controlled violations of the same declared rules; therefore the result verifies implementation correctness for the declared contract. It does not demonstrate that the seven rules detect unknown semantic errors. The aggregate quality score from Equation (1) was 0.9772 because individual records typically violated one of seven dimensions rather than all dimensions simultaneously. Account completeness was the most frequent issue: 8,562 of 100,000 rows failed that rule, largely because the 14 naturally incomplete source rows were replicated during load scaling. The remaining rule pass rates were between 98.659% and 98.872%.

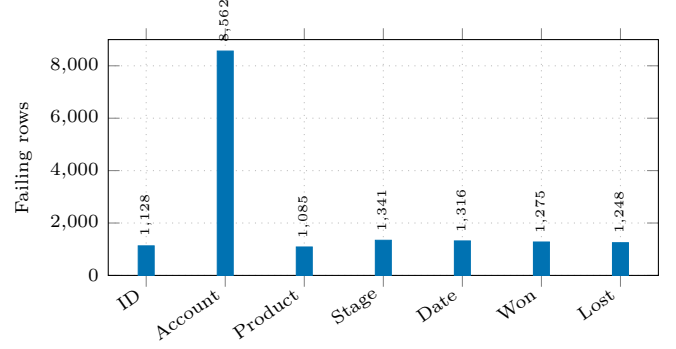


Figure 3: Rule-failure counts in the 100,000-row monitoring workload.

4.2. Model discrimination and calibration

Table 2 reports out-of-fold performance on the 15,000-row grouped benchmark. Random forest produced the highest ranking metrics, with ROC-AUC 0.99998 and PR-AUC 0.99992. XGBoost achieved ROC-AUC 0.99954 and PR-AUC 0.99756. At the F2-optimal threshold of 0.70, XGBoost recalled every review-worthy case in the evaluated subset, with precision 0.9768 and specificity 0.9955. Extremely randomized trees were similarly strong. Logistic regression was materially weaker, with ROC-AUC 0.9280 and F2 0.8066, indicating that the mixed syntax, categorical stage, and value interactions are not well represented by one linear boundary.

These high tree-ensemble scores arise from the benchmark design and should not be generalized to ambiguous master-data or semantic-reference errors. Most injected faults create observable structural changes such as zero string length, negative cycle time, invalid stage category, or stage-value inconsistency. The results therefore show that a learned triage model can recover explicit heterogeneous patterns; they do not establish automatic detection of every real Salesforce data incident.

A more consequential result appears in calibration. XGBoost’s ECE was 0.1968 and its Brier score was 0.0572 despite near-perfect ranking. The class-weighted objective improves sensitivity but distorts the direct probabilistic interpretation of the scores. Accordingly, the deployed quantity should be described as a review-priority score unless post-hoc calibration is performed on independent production labels. This limitation is important because a monitoring dashboard that prints “99% anomaly probability” would be operationally misleading even when the ordering of suspicious records is excellent.

Table 2: Grouped out-of-fold anomaly-prioritization results on 15,000 records.

Model	ROC-AUC	PR-AUC	F1	F2	Precision	Recall	Specificity	ECE
Random Forest	0.99998	0.99992	0.99938	0.99900	1.00000	0.99875	1.00000	0.04904
XGBoost	0.99954	0.99756	0.98828	0.99528	0.97683	1.00000	0.99548	0.19682
Extra Trees	0.99865	0.99564	0.98991	0.98396	1.00000	0.98002	1.00000	0.12621
Logistic	0.92800	0.87212	0.79043	0.80658	0.76489	0.81773	0.95205	0.14442

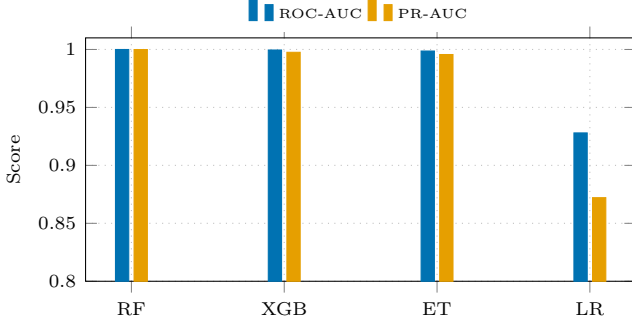


Figure 4: Model comparison under grouped cross-validation.

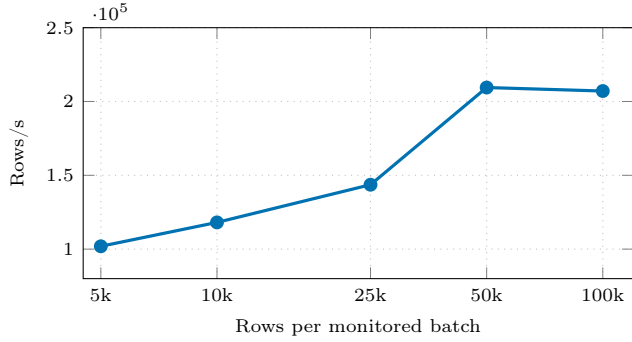


Figure 5: Measured monitoring throughput including validation, feature construction, and XGBoost inference.

4.3. Monitoring throughput

Figure 5 shows measured end-to-end monitoring throughput. The 5,000-row workload required a median 0.0491 s and processed 101,911 rows/s. At 100,000 rows, deterministic validation, feature construction, and XGBoost inference required 0.4829 s, corresponding to 207,092 rows/s. Throughput increased with batch size as fixed Python and pipeline overhead was amortized, reaching 209,454 rows/s at 50,000 rows and remaining near that level at 100,000 rows. These values are not API extraction rates; they measure the local monitoring stage after records are available to the process. Network latency, Salesforce Bulk API behavior, warehouse commit time, and enterprise orchestration are outside this timing boundary.

4.4. SHAP interpretation

The SHAP ranking in Table 3 and Figure 6 is consistent with the benchmark semantics. Account-string length had the highest mean absolute SHAP value (1.1573), reflecting the large number of naturally and artificially missing accounts. The Won-stage indicator ranked second, followed by close value and sales-cycle duration, which are necessary

Table 3: Top-10 XGBoost SHAP features.

Rank	Feature	Mean SHAP
1	Account length	1.1573
2	Stage = Won	0.6742
3	Close value	0.5080
4	Cycle days	0.4047
5	Product length	0.3095
6	ID length	0.2453
7	Stage = Closed-Suspended	0.1626
8	Stage = Engaging	0.0678
9	ID non-alphanumeric count	0.0466
10	Stage = Lost	0.0287

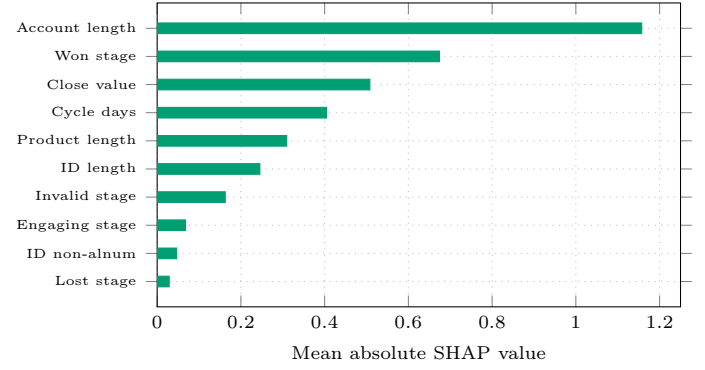


Figure 6: Global SHAP importance for XGBoost.

to represent the contextual value and date violations. Product length, identifier length, the injected invalid stage, and non-alphanumeric identifier count also appeared among the top ten. This alignment provides a useful audit: the model is responding to understandable structure rather than account or agent identity, which were deliberately excluded.

The explanation result also identifies a limitation of the stress protocol: account completeness dominates because the public source already contains missing accounts and volume scaling replicates them. In a production tenant with nearly complete accounts but frequent reference-data drift, the importance ordering could differ substantially. Consequently, feature importance should be monitored over time rather than treated as a fixed domain truth.

4.5. Batch-level quality variability

The 100,000-row workload produced 100 consecutive 1,000-row batch summaries. Because the controlled faults were distributed under a fixed random seed rather than concentrated into a single incident window, the observed batch anomaly rate varied from 12.9% to 19.0%, with a mean of 15.955% and standard deviation of 1.139 percentage points. The corresponding data-quality score varied from 0.9729

to 0.9816. Batch 16 was the weakest observed batch, with a 19.0% review rate and $Q = 0.9729$; batch 53 was the strongest, with a 12.9% review rate and $Q = 0.9816$.

These values illustrate why an enterprise dashboard should retain both row counts and normalized quality dimensions. A batch with 190 suspicious rows is operationally different from one with 129 even though both may contain the same set of rule types. At the same time, $Q(B)$ should not be interpreted as a universal business-risk score: because Eq. (1) weights the seven rules equally, one severe stage-domain failure contributes the same mathematical amount as one missing account. Production deployments should therefore add rule-specific severity weights only after governance owners define the business consequences of each violation. This paper keeps equal weights to avoid inventing organization-specific costs.

For an illustrative, non-optimized service threshold of $Q < 0.975$, eight of the 100 benchmark batches would have generated a quality-degradation alert. Raising the threshold to 0.976 would flag 19 batches. This sensitivity shows that alert volume can change materially with a seemingly small threshold adjustment; therefore threshold tuning should use historical incident labels, acceptable operator workload, and the cost of missed quality regressions. The result also reinforces the role of Eq. (5): persistence logic can reduce reactions to isolated noisy batches without weakening deterministic row quarantine.

The monitored batch stream is intentionally stationary apart from sampling variation, so it does not constitute a concept-drift benchmark. A realistic deployment would encounter seasonal sales behavior, changing opportunity stages, newly introduced products, and administrative schema changes. Under such conditions, a shift in SHAP attribution or score distribution could be an early diagnostic signal, but it would still require confirmation against rule events and metadata change logs before being labeled as data degradation. This layered interpretation prevents ordinary business evolution from being confused with pipeline failure.

4.6. Operational implications

The combined evidence supports a conservative architecture. Rule evaluation remains the contractual gate because it provides exact, testable semantics for known conditions. XGBoost is useful for prioritizing human attention, combining heterogeneous patterns, and producing a compact review score, but its poor calibration prohibits interpreting the raw score as a defect probability. Execution measurements show that the local monitoring stage can be inexpensive relative to typical network-bound ETL operations, although the experiment does not include Salesforce API limits or cloud-warehouse commit latency. The framework is therefore most defensible as a quality-control layer inserted between normalized staging and warehouse loading, with quality events persisted for longitudinal analysis.

5. Conclusion

This paper presented an intelligent ETL monitoring and data-quality validation framework for Salesforce-compatible enterprise integration. The design separates deterministic

acceptance rules from an explainable learned prioritization branch and records both quality evidence and execution telemetry. Using a public 201-row CRM snapshot, 14 naturally missing-account records were preserved and seven additional fault classes were introduced under a fixed, reproducible protocol. The resulting 100,000-row workload contained 15,955 review-worthy records. The declared rule contract recovered all benchmark-defined violations, while XGBoost achieved grouped out-of-fold ROC-AUC 0.9995, PR-AUC 0.9976, F2 0.9953, and recall 1.0000 on the 15,000-row model subset. End-to-end monitoring of 100,000 records required 0.483 s and reached 207,092 rows/s in the reported environment. SHAP attributions were dominated by account completeness, Won-stage context, close value, and cycle duration.

The strongest limitation is also operationally important: XGBoost ECE was 0.1968 despite excellent ranking performance. The model score should therefore be treated as review priority, not calibrated defect probability. In addition, the controlled faults represent explicit structural and business-rule violations; they do not cover ambiguous entity resolution, stale master data, cross-system semantic disagreement, or schema evolution. Future work should evaluate multi-day Salesforce extracts with real incident labels, incorporate drift detection and calibrated probabilities, measure API-to-warehouse latency under cloud orchestration, and extend the event model to automatically identify which source object or transformation introduced a defect.

Declarations and ethical statements

Conflict of interest: The author declares that there is no conflict of interest.

Funding statement: The author declares that no specific funding was received for this research.

Artificial Intelligence usage statement: During the preparation of this manuscript, the author utilized ChatGPT solely for language refinement and grammatical corrections. The author carefully reviewed and revised the generated content and take full responsibility for the accuracy, integrity, and originality of the final manuscript.

Availability of data and materials: The data and/or materials that support the findings of this study are available from the corresponding author upon reasonable request.

CRedit authorship contribution statement

Rama Krishna Inmmuri: Conceptualization, Investigation, Methodology & Original draft writing.

Publisher's note

Krrish Scientific Publications Pvt. Ltd. and the Journal of Computing and Data Technology remain neutral with regard to jurisdictional claims in published maps and institutional affiliations.

References

- [1] Wang RY, Strong DM. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*. 1996;12(4):5–33. Available from: <https://doi.org/10.1080/07421222.1996.11518099>
- [2] Pipino LL, Lee YW, Wang RY. Data quality assessment. *Communications of the ACM*. 2002;45(4):211–218. Available from: <https://doi.org/10.1145/505248.506010>
- [3] Batini C, Cappiello C, Francalanci C, Maurino A. Methodologies for data quality assessment and improvement. *ACM Computing Surveys*. 2009;41(3):1–52. Available from: <https://doi.org/10.1145/1541880.1541883>
- [4] Batini C, Scannapieco M. *Data Quality: Concepts, Methodologies and Techniques*. Berlin, Germany: Springer; 2006. Available from: <https://doi.org/10.1007/3-540-33173-5>
- [5] Vassiliadis P. A survey of extract-transform-load technology. *International Journal of Data Warehousing and Mining*. 2009;5(3):1–27. Available from: <https://doi.org/10.4018/jdwm.2009070101>
- [6] Ali SMF, Wrembel R. From conceptual design to performance optimization of ETL workflows: Current state of research and open problems. *The VLDB Journal*. 2017;26:777–801. Available from: <https://doi.org/10.1007/s00778-017-0477-2>
- [7] Souibgui M, Atigui F, Zammali S, Si-Said Cherfi S, Ben Yahia S. Data quality in ETL process: A preliminary study. *Procedia Computer Science*. 2019;159:676–687. Available from: <https://doi.org/10.1016/j.procs.2019.09.223>
- [8] Abedjan Z, Chu X, Deng D, Fernandez RC, Ilyas IF, Ouzzani M, Papotti P, Stonebraker M, Tang N. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*. 2016;9(12):993–1004. Available from: <https://doi.org/10.14778/2994509.2994518>
- [9] Dallachiesa M, Ebaid A, Eldawy A, Elmagarmid AK, Ilyas IF, Ouzzani M, Tang N. NADEEF: A commodity data cleaning system. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013. p. 541–552. Available from: <https://doi.org/10.1145/2463676.2465327>
- [10] Khayyat Z, Ilyas IF, Jindal A, Madden S, Ouzzani M, Papotti P, Quiane-Ruiz JA, Tang N, Yin S. BigDansing: A system for big data cleansing. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 2015. p. 1215–1230. Available from: <https://doi.org/10.1145/2723372.2747646>
- [11] Krishnan S, Wang J, Wu E, Franklin MJ, Goldberg K. ActiveClean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*. 2016;9(12):948–959. Available from: <https://doi.org/10.14778/2994509.2994514>
- [12] Rekatsinas T, Chu X, Ilyas IF, Re C. HoloClean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*. 2017;10(11):1190–1201. Available from: <https://doi.org/10.14778/3137628.3137631>
- [13] Chu X, Morcos J, Ilyas IF, Ouzzani M, Papotti P, Tang N, Ye Y. KATARA: Reliable data cleaning with knowledge bases and crowdsourcing. *Proceedings of the VLDB Endowment*. 2015;8(12):1952–1955. Available from: <https://doi.org/10.14778/2824032.2824109>
- [14] Mahdavi M, Abedjan Z, Fernandez RC, Madden S, Ouzzani M, Stonebraker M, Tang N. Raha: A configuration-free error detection system. In: *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*. 2019. p. 865–882. Available from: <https://doi.org/10.1145/3299869.3324956>
- [15] Heidari A, McGrath J, Ilyas IF, Rekatsinas T. HoloDetect: Few-shot learning for error detection. In: *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*. 2019. p. 829–846. Available from: <https://doi.org/10.1145/3299869.3319888>
- [16] Chandola V, Banerjee A, Kumar V. Anomaly detection: A survey. *ACM Computing Surveys*. 2009;41(3):15. Available from: <https://doi.org/10.1145/1541880.1541882>
- [17] Breunig MM, Kriegel HP, Ng RT, Sander J. LOF: Identifying density-based local outliers. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. 2000. p. 93–104. Available from: <https://doi.org/10.1145/342009.335388>
- [18] Liu FT, Ting KM, Zhou ZH. Isolation forest. In: *2008 Eighth IEEE International Conference on Data Mining*. 2008. p. 413–422. Available from: <https://doi.org/10.1109/ICDM.2008.17>
- [19] 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*. 2016. p. 785–794. Available from: <https://doi.org/10.1145/2939672.2939785>
- [20] Lundberg SM, Erion G, Chen H, DeGrave A, Prutkin JM, Nair B, Katz R, Himmelfarb J, Bansal N, Lee SI. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*. 2020;2:56–67. Available from: <https://doi.org/10.1038/s42256-019-0138-9>
- [21] Breiman L. Random forests. *Machine Learning*. 2001;45:5–32. Available from: <https://doi.org/10.1023/A:1010933404324>
- [22] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*. 2002;16:321–357. Available from: <https://doi.org/10.1613/jair.953>
- [23] Fawcett T. An introduction to ROC analysis. *Pattern Recognition Letters*. 2006;27(8):861–874. Available from: <https://doi.org/10.1016/j.patrec.2005.10.010>