Self-Healing Data Pipelines with Generative AI

Deterministic Governance for Multimodal Labeling, Data Quality Repair, and ML-Ready Feature Stores

Prepared for public distribution

Date: 2026-01-14

Abstract

Abstract. Noisy real-world data pipelines frequently fail not because modeling is hard, but because upstream data is inconsistent, incomplete, and weakly supervised. This thesis presents a practical architecture for *self-healing databases*: an operational data store that continuously repairs itself, labels missing attributes, and converges toward a canonical, training-ready representation.

The key contribution is a **two-node generative AI integration**. The first node operates over text fields (titles, descriptions, structured attributes) to perform spam triage, identity normalization, evidence-gated enrichment, and controlled re-labeling. The second node operates over images to extract orthogonal signals (visible damage, accessories, packaging, body color) and, when confidence is high, propose identity repairs that are not reliably recoverable from text alone.

Crucially, generative AI is not treated as an unconstrained oracle. Every model output is mediated by deterministic governance: closed taxonomies, strict JSON schemas, confidence thresholds, evidence requirements, diff-only patches, idempotent sentinel flags, and append-only audit logs. These mechanisms transform probabilistic model outputs into safe database mutations.

The resulting dataset supports downstream machine-learning systems such as gradient-boosted decision trees by reducing label noise, increasing feature coverage, and stabilizing targets via post-event snapshots. The architecture is model-agnostic, domain-agnostic, and designed to be cost-efficient through batching, model routing, and at-most-once processing contracts.

Executive Summary

This thesis describes an operational pattern for transforming noisy marketplace-style listings into a continuously improving, ML-ready dataset. The system combines generative AI with deterministic controls to ensure correctness, cost efficiency, and reproducibility.

- **Two-node generative AI design:** independent text and vision labeling nodes with a shared governance layer and a convergent feature store.
- **Deterministic safety envelope:** every AI proposal must satisfy validators (schema, evidence, confidence thresholds, closed vocabularies) before it is persisted.
- **Truth hierarchy:** hard rules and post-event snapshots override mutable operational fields; AI outputs are advisory unless validated.
- **Idempotent execution:** sentinel flags and version tags enable safe re-runs, controlled re-labeling, and at-most-once processing for expensive steps.
- **Auditable mutations:** append-only JSON audit entries capture what changed, why it changed, and which policy/model version produced the change.
- **Cost engineering:** batching, downsampling, small-model routing, and diff-only patching reduce inference spend while maintaining label quality.
- **Downstream impact:** improved feature coverage and stabilized targets directly benefit gradient-boosted decision trees and other supervised learners.

Scope and sanitization. The discussion is intentionally generalized. Specific data sources, proprietary identifiers, URLs, credentials, and production endpoints are excluded. All examples use neutral terms such as *listing_id*, *operational database*, and *snapshot anchors*.

Table of Contents

List of Figures and Tables

Figures are designed to be self-contained and intentionally avoid any proprietary identifiers. All diagrams use generalized entity names (listing, snapshot, image assets, feature store).

1. Introduction

Most supervised machine-learning projects inherit a hidden assumption: the training data is already correct, consistently structured, and aligned with the prediction task. In practice, this assumption rarely holds. Real-world pipelines ingest heterogeneous content (structured fields, free-form text, and images) produced by many independent actors. The resulting dataset contains spam, duplicates, missing attributes, contradictory signals, and drift over time.

Traditional data cleaning is often implemented as a growing collection of brittle rules. These rules are expensive to maintain, difficult to generalize across domains, and frequently fail in the tail of language and image variation.

This thesis adopts a data-centric view: improving model performance is primarily achieved by improving the quality and consistency of the data substrate. We propose a self-healing database architecture in which data quality repair is a first-class, continuous process rather than a one-off preprocessing step.

Generative AI provides the semantic capacity to interpret unstructured content, but its outputs are probabilistic. Therefore, the central challenge is not how to call a model, but how to convert model outputs into safe and deterministic database updates.

Thesis statement. A robust self-healing pipeline can be constructed by integrating generative AI at two points (text and vision) and surrounding it with deterministic governance: validators, invariants, evidence requirements, and auditability. This combination yields ML-ready features and targets with lower noise and higher coverage than raw ingestion alone.

2. Data Quality Problem Setting

We consider a common class of datasets: user-generated product listings containing structured attributes (price, location), free-form text (title, description), and a set of images. The dataset is continuously appended as new listings arrive and as existing listings change state (e.g., edited, removed, sold).

The pipeline is designed to address these failure modes:

- **Off-domain and spam:** non-target items, solicitation posts, service offers, and accessory-only listings mixed with primary products.
- **Identity ambiguity:** inconsistent product naming, missing model identifiers, or incorrect generation/variant fields.
- **Duplicates and fragmentation:** multiple rows representing the same real-world listing due to re-scrapes, edits, or inconsistent parsing.
- **Missingness:** key attributes present only in text or images (storage, lock status, accessories, battery health) but absent from structured fields.
- **Contradictory signals:** text claims that conflict with photographs; condition labels that conflict with observed wear; or stale attributes after edits.
- **Temporal instability:** prices and descriptions change, while downstream modeling requires stable targets and consistent definitions.

A key observation is that many high-value attributes are not reliably available from a single modality. Battery health may be mentioned in text, embedded in a screenshot, or omitted entirely; visible damage may be photographed but not described; and accessories are often shown visually rather than enumerated in text.

3. System Architecture Overview

The system is organized as an eventually consistent labeling mesh. Newly ingested rows enter an operational database as raw records. Two independent generative AI nodes then propose labels and repairs: a **text node** operating over titles and descriptions, and a **vision node** operating over images.

Both nodes write into dedicated feature stores and/or propose patches to canonical operational fields. A deterministic governance layer validates proposals, applies conflict resolution policies, and ensures that every mutation is auditable and idempotent.

A separate snapshot mechanism captures post-event ground truth (e.g., after an item is no longer live), creating stable anchors for reconciliation. Downstream ML consumes curated feature views produced from the merged operational and feature-store data.

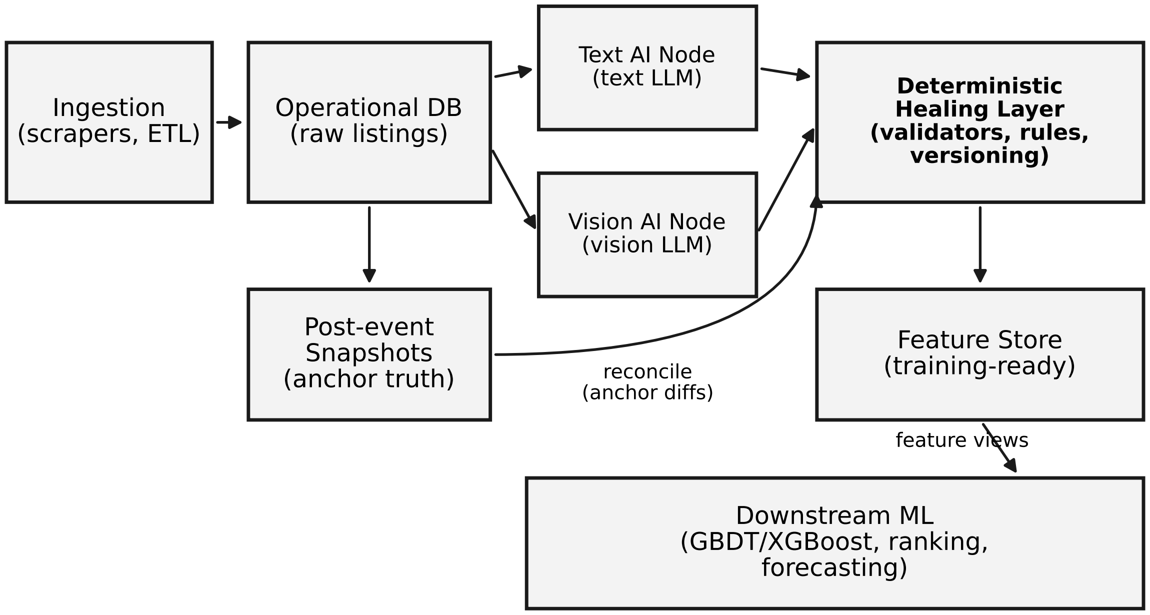

Ingestion (scrapers, ETL)
Operational DB (raw listings)
Text AI Node (text LLM)
Vision AI Node (vision LLM)
Deterministic Healing Layer (validators, rules, versioning)
Post-event Snapshots (anchor truth)
reconcile (anchor diffs)
Feature Store (training-ready)
feature views
Downstream ML (GBDT/XGBoost, ranking, forecasting)

Figure 1. Two-node Generative AI pipeline with deterministic governance.

4. Truth Hierarchy and Conflict Resolution

A self-healing database requires a principled definition of truth. Raw operational fields are mutable and may be stale or incorrect. Generative AI outputs are probabilistic and may be wrong. The system therefore defines an explicit **truth hierarchy** that governs how conflicts are resolved.

Hard rules encode invariants (for example, closed-range constraints, mutually exclusive states, or deterministic clamps for extreme conditions). **Snapshot anchors** provide higher-trust observations captured after a controlled delay, reducing the impact of transient edits. **Deterministic parsers** extract high-precision facts from text or metadata. Finally, **LLM proposals** fill gaps and interpret ambiguous cases, but only after validation.

Figure 2. Truth hierarchy and conflict resolution

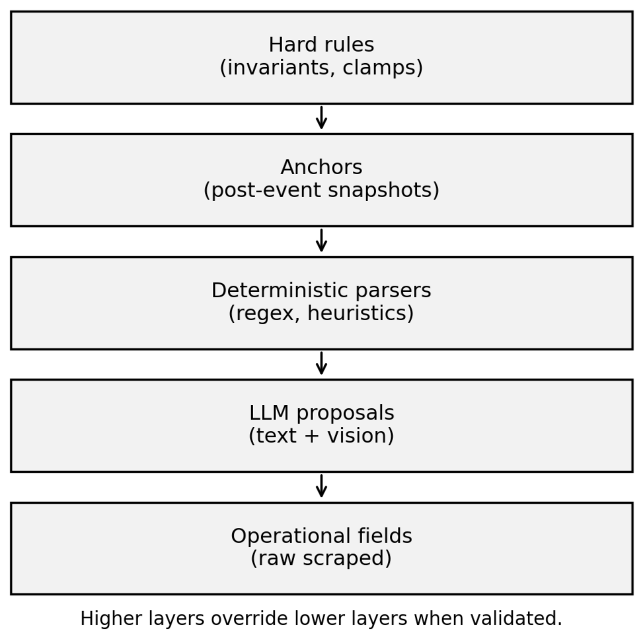


Figure 2. Truth hierarchy and conflict resolution. Higher layers override lower layers when validated.

This hierarchy enables deterministic updates: when a higher layer changes (e.g., a new snapshot arrives), the system can invalidate dependent labels and trigger re-labeling. This is a critical mechanism for convergence, described later as dependency invalidation.

5. Data Model and Feature Stores

The architecture separates **operational records** (the mutable, queryable representation of the listing) from **derived feature stores** (structured outputs produced by labeling modules). This separation prevents accidental overwrites of raw fields and makes re-labeling safe.

A minimal generalized model includes: (i) an operational listings table, (ii) a snapshot table that stores post-event anchors, (iii) an image asset table that stores pointers to downloaded images and captions, and (iv) one or more feature stores keyed by listing_id and a version tag. Feature stores are appendable and can be recomputed without re-scraping.

Figure 4. Generalized data model: operational tables and feature store

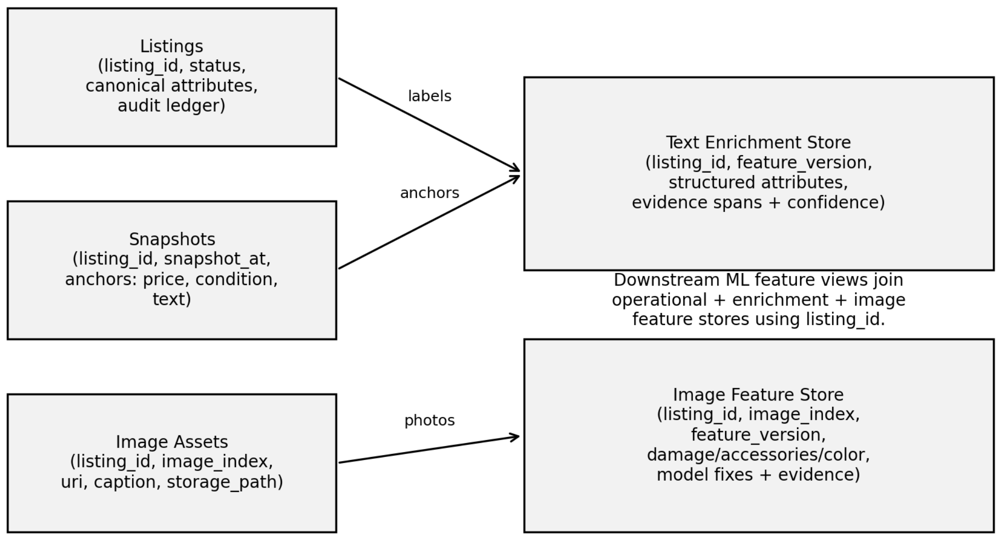

Listings (listing_id, status, canonical attributes, audit ledger)
Snapshots (listing_id, snapshot_at, anchors: price, condition, text)
Image Assets (listing_id, image_index, uri, caption, storage_path)
labels
anchors
photos
Text Enrichment Store (listing_id, feature_version, structured attributes, evidence spans + confidence)
Downstream ML feature views join operational + enrichment + image feature stores using listing_id.
Image Feature Store (listing_id, image_index, feature_version, damage/accessories/color, model fixes + evidence)

Figure 4. Generalized data model: operational tables and feature store.

Every derived table includes explicit provenance fields such as *feature_version*, *model_version*, timestamps, and (when applicable) evidence payloads. Downstream ML consumes curated feature views by joining these stores on listing_id and selecting the appropriate versions.

6. Text Node I: Triage and Identity Repair

The first text-based module performs **triage**: it classifies each new or changed row into a small set of operational categories (in-scope listing, off-domain, solicitation, accessory-only, duplicate candidate). The objective is to prevent downstream labeling and modeling from consuming irrelevant or harmful rows.

Generative AI is used because spam and off-domain content often depends on semantics, not simple keywords. However, the system constrains the model through a strict output schema and deterministic repair rules. A typical output is a JSON object containing: a spam label (or null), an inferred product type, and optional identity repairs such as normalized model name or corrected generation.

Identity repairs are deliberately conservative. For example, a generation correction may be allowed only when the same generation is mentioned unambiguously in multiple independent text fields, and when the inferred correction does not conflict with structured attributes. When the evidence is weak or contradictory, the module must return null (no fix).

Algorithm sketch: Safe identity repair in the text node
1. Select candidate rows where critical fields are missing or out-of-range.
2. Call the LLM with a strict JSON schema and explicit rejection rules.
3. Validate: (a) schema, (b) allowed label set, (c) evidence consistency, (d) no contradiction with immutable invariants.
4. Apply diff-only updates (only fields that change) and append an audit entry.
5. Stamp the row with a version tag to prevent reprocessing.

The output is a cleaner operational dataset with fewer off-domain rows and improved identity consistency. This early filter has compounding benefits: later modules run on fewer rows, spend fewer tokens, and make fewer mistakes.

7. Text Node II: Evidence-Gated Enrichment

After triage, the pipeline enriches in-scope rows by extracting structured attributes that are present in text but absent or unreliable in scraped columns. Examples include accessory mentions, lock status, repair parts, shipping terms, warranty language, and other attributes that are costly to encode as hand-written rules.

A core design choice is that the LLM must not only output values, but also **justify** them with evidence. For each emitted label, the model returns a confidence score and one or more evidence spans referencing the exact region of text that supports the label. Deterministic validators check that evidence spans are in-bounds, confidence exceeds a threshold, and that the label is within a closed taxonomy.

Enrichment writes into a separate feature store keyed by listing_id and a feature version. This prevents accidental loss of raw fields and allows repeated enrichment passes as policies improve. Upserts use conservative merge rules (for instance, never overwriting a high-trust scraped value with a lower-trust inferred value).

Evidence gating converts probabilistic extraction into deterministic data:

- Reject labels without explicit evidence spans.
- Reject labels below the confidence threshold.
- Reject values outside closed vocabularies (e.g., enumerated sale modes).
- Preserve higher-trust fields by design (scraped values can dominate).

This mechanism is a practical alternative to manual labeling: it produces a large volume of structured labels while controlling hallucination risk through deterministic checks. The resulting enrichment store acts as a stable interface for downstream feature engineering.

8. Snapshot Anchors and Reconciliation

Many of the most important modeling targets are only reliable after an item transitions out of the live state. For example, a final sold price or definitive condition label may not be stable during the listing's active lifecycle. The pipeline therefore captures **post-event snapshots** after a configurable delay, creating a stable anchor record.

Snapshot capture is intentionally redundant: it uses multiple extraction strategies (structured embedded metadata, JSON-LD, HTML fallbacks) to recover text and key fields. Snapshot rows include diagnostic metadata (HTTP status, parsing outcomes) so that failed snapshots can be excluded from reconciliation.

Once a snapshot is available, the pipeline performs **diff-only reconciliation**. Only rows with meaningful differences between snapshot fields and operational fields are updated. Every reconciliation patch includes a provenance entry in an append-only audit ledger.

Reconciliation also enables dependency invalidation. When a high-level truth (e.g., condition score) changes, dependent labels (e.g., damage severity) are cleared or queued for re-labeling. This prevents stale derived labels from persisting after upstream corrections.

Dependency invalidation. If an upstream attribute changes beyond a tolerance (EPS), downstream labels that depend on it are invalidated by clearing their version tag. A later module reprocesses only invalidated rows, restoring convergence without expensive full re-labeling.

9. Deduplication and Canonicalization

Duplicate records arise naturally in continuously ingested datasets: the same listing may be scraped multiple times, parsed inconsistently, or inserted under multiple candidate identities. If unaddressed, duplicates inflate counts, bias training, and contaminate feature distributions.

The deduplication module groups rows by listing_id and uses a combination of deterministic heuristics and a constrained LLM decision to select a canonical record. To reduce the impact of stale content, the module may re-verify a subset of fields (e.g., title and description) from the current page representation before deciding.

The decision output is again constrained: a set of indices to keep, and an optional bundle indicator when multiple physical items are explicitly sold together. Non-kept rows are marked as duplicates (a spam-like label) to exclude them from downstream pipelines. When safe, canonical status and outcome fields are merged using a monotone ordering (e.g., sold dominates live).

Canonicalization is not simply data deletion. It is an explicit policy layer that preserves provenance while producing a single consistent representation for modeling. All actions are logged to the audit ledger, enabling retrospective debugging and policy iteration.

10. Battery Refit and At-Most-Once Processing

Certain attributes are both high-signal for price and notoriously inconsistent in raw data. Battery health is a representative example: it can be written in many linguistic forms, confused with charge level, or omitted.

The battery refit module uses a narrowly scoped extraction prompt: it returns a single percentage only when the text clearly refers to battery health or maximum capacity. Ambiguous numbers (e.g., charging limits) are explicitly rejected. Outputs are normalized (rounding decimals, mapping ranges to lower bounds) and clamped to valid ranges.

To improve reliability and to support evaluation, the module can evaluate two sources per listing: the operational text and the post-event snapshot text. Final writes follow a deterministic precedence rule (e.g., snapshot-first). The module writes the chosen value to the operational record and optionally backfills the snapshot row.

Cost control and idempotency are achieved with an **at-most-once sentinel**: each snapshot row is processed at most once by stamping a fetched marker. This converts an expensive LLM step into a bounded, monotone queue that naturally drains as new snapshots arrive.

In addition to the value itself, the module appends a structured log entry describing the source chosen, evidence text, and prior values. These logs support quality dashboards (coverage, disagreement rates) and targeted sampling for human review.

11. Damage Labeling: Deterministic + Generative

Damage labeling illustrates the central design principle: combine deterministic rules with generative interpretation. Some states are unambiguous and should not require model inference. For example, items explicitly labeled as new can be clamped to 'no visible damage' and bypass the LLM.

For the remaining cases, a constrained LLM acts as a decision function that maps text signals into a small severity scale (e.g., 0-3) and a binary indicator. The prompt encodes an explicit priority ladder: structural cracks and functional faults dominate cosmetic wear; non-original parts trigger elevated severity; and low battery health can clamp the severity when policy requires.

Every decision is persisted as both structured columns (binary, severity, reason) and as a full JSON payload containing evidence and policy version tags. This makes the label auditable and reproducible, and it supports later policy changes through versioned re-labeling.

Multimodal independence. A critical architectural choice is that image-based damage assessment is computed independently from text-based damage assessment. This creates an internal consistency check: disagreements become valuable signals for fraud detection, seller reliability scoring, or additional review sampling.

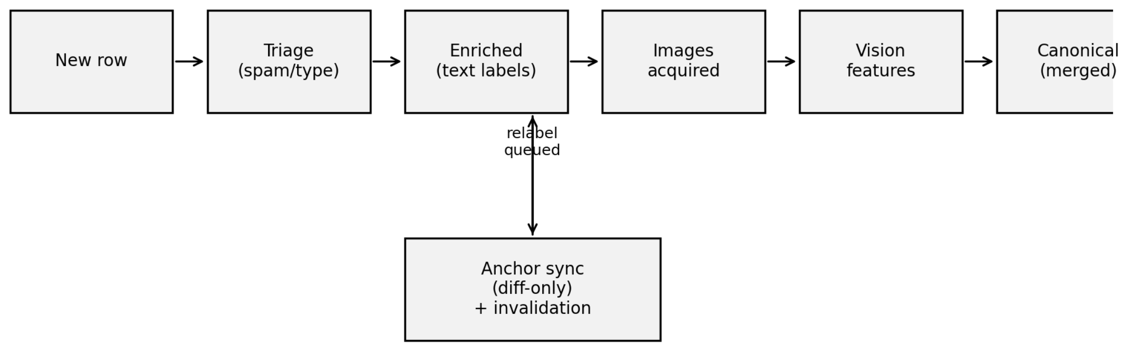
12. Vision Node Overview and Convergence

The second generative AI node operates over images. Images are first treated as first-class data: the pipeline downloads and indexes them per listing_id and image_index, storing both the source URI and any per-image caption text.

Vision labeling is split into multiple inexpensive passes that each own a disjoint set of feature columns. Examples include visible damage, accessory detection, packaging state, battery-health screenshots, and body color. Each pass is idempotent: it sets a done flag at the listing scope to prevent repeated spend.

The vision node may also propose high-confidence identity repairs when photographs contradict the operational fields. Such repairs are only applied under strict confidence and validation thresholds and are recorded in an explicit audit table.

Figure 3. Convergent labeling state machine (idempotent steps)



Sentinel flags prevent reprocessing; version tags enable controlled re-labeling.

Figure 3. Convergent labeling state machine (idempotent steps).

13. Vision Damage and Photo-Derived Signals

Vision-based damage labeling produces signals that are either impossible or unreliable to infer from text alone. Examples include visible cracks, scuffs, dents, lens damage, and the presence of a screen protector.

The vision damage module is explicitly conservative. It must label only what is visible, distinguish actual damage from reflections or protective films, and report uncertainty via bounded scales (e.g., a 0-10 visibility score) rather than free-form text.

In addition to damage, the module extracts quality-of-evidence features: photo quality, background cleanliness, and a stock-photo likelihood indicator. These features are valuable not only for pricing but also for risk modeling, as low-quality photos often correlate with higher uncertainty and lower buyer confidence.

Cost efficiency. The vision pipeline reduces cost by downsampling images to a bounded resolution, batching multiple images into a single request, and routing tasks to smaller models when the decision complexity is low. Idempotent done flags prevent repeated calls on unchanged listings.

14. Vision Accessories and Screenshot Extraction

Accessory presence and packaging state can materially affect price and time-to-sale, yet they are inconsistently described in text. Vision labeling detects accessories such as chargers, cables, earbuds, cases, and original packaging, and can estimate counts when multiple items are visible.

A particularly high-value capability is **screenshot extraction**. Sellers often include screenshots of device settings that show battery health or other diagnostics. These values may not be present in text and may be more trustworthy than free-form claims. The vision module recognizes these screenshots and extracts the relevant numeric values.

The extracted accessory and screenshot features are stored per image and then aggregated per listing (e.g., max battery-health observed across images, presence flags). Because these features are photo-derived, they remain available even when textual descriptions are sparse.

The pipeline sanitizes and validates extracted numbers (range checks, nulling implausible values) before persistence. This is the same governance pattern used in the text node: probabilistic extraction followed by deterministic acceptance criteria.

15. Vision Color and High-Confidence Identity Repair

Color and product variant are frequent sources of inconsistency. Text may omit color, use non-standard names, or contain contradictory descriptions. Vision labeling provides a direct signal by classifying body color from images.

To prevent taxonomy drift, the system enforces **closed vocabularies**: each product family and generation has an allowed set of colors. If the model outputs a color outside the allowed set, the label is rejected rather than coerced.

Beyond color, the vision node can propose identity repairs (e.g., variant or generation). These repairs are only applied when confidence is high and when the inferred identity is consistent across multiple images. When applied, the repair is transactional: dependent records in the image-asset table and feature store are moved together to preserve referential integrity.

Every identity repair is written to a dedicated audit log that records the prior state, the new state, the confidence, and the evidence rationale. This prevents silent data corruption and enables later review.

16. Deterministic Governance: Validators and Invariants

The distinguishing feature of the system is not the presence of LLM calls, but the deterministic envelope that surrounds them. Governance mechanisms convert uncertain model outputs into safe database mutations and provide reproducibility under policy evolution.

Table 1 summarizes the primary governance controls used throughout the pipeline.

Control	Purpose	Example
Strict JSON schema	Reject malformed outputs; enforce types	Model must return {label, confidence, evidence}
Closed taxonomies	Prevent drift and invalid categories	Color must be in allowed set; else null
Evidence requirement	Reduce hallucinations; enable audits	Text span offsets must be in-bounds
Confidence thresholds	Gate low-certainty labels	Accept only if confidence >= τ
Range and sanity checks	Prevent invalid numeric values	Clamp battery health to [20,100]
Diff-only patching	Avoid churn and preserve stability	Update only changed columns
Sentinel flags (done/fetched)	At-most-once processing and cost control	Skip rows already labeled
Version tags	Controlled re-labeling and reproducibility	Store policy_version per module
Append-only audit ledger	Traceability of every mutation	Log before/after and rationale
Transactional identity repair	Maintain referential integrity	Move dependent rows with identity fix
Source precedence rules	Deterministic conflict resolution	Snapshot-first when both available
Dependency invalidation	Prevent stale downstream labels	Clear damage_version if condition changes

Table 1. Deterministic governance controls for safe database mutation.

17. Cost Engineering and Throughput Control

A practical self-healing system must be economically sustainable. The pipeline achieves cost efficiency through architectural, algorithmic, and operational strategies.

Batching and amortization. Multiple items (or multiple images per item) are processed in a single request where possible, amortizing prompt overhead. **Model routing.** Tasks that require fine-grained reasoning (e.g., nuanced damage) can use a stronger model, while simple classifiers (e.g., accessory presence) are routed to smaller models.

Resolution bounds. Images are downsampled to a fixed maximum edge length to reduce tokenization and compute while preserving salient details. **Idempotency gates.** Sentinel flags prevent repeated processing of the same snapshot or listing.

Throughput control. Concurrency and request-per-second budgets are treated as first-class parameters. Backoff, jitter, and retry policies are used to maintain stability under rate limits.

The combined effect is that generative AI is applied surgically: only to rows that need it, only once per processing epoch, and only with outputs that survive deterministic validation. This enables large-scale labeling without uncontrolled operational cost.

18. Feature Views for Gradient-Boosted Trees

The output of the self-healing pipeline is not a single table, but a set of curated feature views. A feature view is a deterministic join of operational fields and derived stores, selecting a consistent version of each feature family.

Gradient-boosted decision trees (GBDTs), including XGBoost-style learners, are well-suited to this setting: they handle heterogeneous feature types, non-linear interactions, and missing values. However, their performance is highly sensitive to label noise and systematic missingness.

The pipeline improves downstream learning in three ways: (i) it reduces spam and duplicates, preventing distributional bias; (ii) it increases feature coverage via text and vision enrichment; and (iii) it stabilizes targets using snapshot anchors and reconciliation.

Training-only signals. Some information becomes available only after an event (e.g., final outcome) or through internal processing (e.g., post-event snapshots). Such signals can be used to improve training labels, to calibrate uncertainty estimates, or to train auxiliary models for operations planning. Feature governance should explicitly separate: (a) inference-available features, and (b) training-only features used for supervision or evaluation.

19. Evaluation, Monitoring, and Operations

Because labeling is automated, evaluation and monitoring are mandatory. The system treats label quality as an observable, continuously measured quantity rather than an assumption.

Coverage metrics. Track how often each feature is populated (by source and by version), and how coverage changes over time. **Agreement metrics.** Compare independent labelers (text vs vision) and measure disagreement rates; spikes indicate drift or upstream scraping changes. **Stability metrics.** Monitor diff-only reconciliation rates; sudden increases may indicate parser regressions or external platform changes.

Human sampling. The audit ledger enables targeted review: sample rows with low confidence, high disagreement, or large deltas after reconciliation. This supports rapid policy iteration without labeling the entire dataset.

Security and privacy. Operational deployments should isolate secrets from code, avoid logging sensitive content, and minimize retention of raw assets. Public releases should generalize identifiers and remove any credentials, endpoints, and data-source names, while preserving the technical substance of the approach.

References

1. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

2. Sculley, D., Holt, G., Golovin, D., et al. (2015). Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems (NIPS) Workshop.

3. Ratner, A., De Sa, C., Wu, S., Selsam, D., & Ré, C. (2017). Data programming: Creating large training sets, quickly. Advances in Neural Information Processing Systems (NeurIPS).

4. Ratner, A., Bach, S., Ehrenberg, H., Fries, J., Wu, S., & Ré, C. (2020). Snorkel: Rapid training data creation with weak supervision. The VLDB Journal.

5. Northcutt, C. G., Jiang, L., & Chuang, I. L. (2021). Confident learning: Estimating uncertainty in dataset labels. Journal of Artificial Intelligence Research.

6. Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., & Lawrence, N. D. (2008). Dataset shift in machine learning. MIT Press.

7. Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems (NeurIPS).

8. Wei, J., Wang, X., Schuurmans, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems (NeurIPS).

9. Ng, A. (2021). Data-centric AI: Improving model performance by improving data. (Practitioner perspective; widely cited in industry).

Note: The system described in this thesis is domain-agnostic and can be applied to other datasets that combine unstructured text, images, and temporal state changes. References are provided for foundational concepts (weak supervision, dataset shift, GBDT modeling, and LLM prompting), not for any proprietary implementation.