**Microsoft Fabric – Accounts Deduplication Pipeline Documentation**

Version 2025-09-26 20:50 UTC

# 1. Purpose & Audience

This document is the authoritative business + engineering guide for the Accounts Deduplication pipeline in Microsoft Fabric. It covers the business rationale, user flows, technical architecture, stage-by-stage implementation, configuration, operational runbook, data tables, and audit controls. It is intended for data stewards, analytics users, ML/engineering teams, and platform administrators.

# 2. Executive Summary (Business Overview & Objectives)

**Goal**

Create a single, canonical record for each real‑world customer account. Every raw record (member) maps to exactly one **master** record (entity). The pipeline combines:

1. deterministic rules (exact/"perfect" matches),
2. high‑confidence vector similarity (auto ≥ **T\_AUTO**, default 0.95),
3. LLM adjudication for the gray band (≥ **T\_LLM\_LOW**, default 0.70 and < T\_AUTO), and
4. optional human review for low‑confidence or ambiguous pairs.

Only after proposals are validated (LLM/Reviewer/Admin) do we **apply merges**. All decisions are logged for audit and can be replayed at any time. Historical decisions and applied mappings are re‑used on subsequent runs (CDC) so we never re‑work what has already been finalized.

**Overview**

* **Normalize & group** obvious duplicates (exact/near‑exact names).
* **Compare masters only** using embeddings to find likely duplicates across the catalog.
* **Route each pair**: auto‑merge if very similar; send to the LLM when uncertain; ignore if clearly different.
* **Ask for help** only when needed:
  + The LLM decides **YES / NO / NEEDS\_CONFIRMATION**;
  + borderline or “needs confirmation” pairs go into a **review queue**;
  + admins can **approve or reject** any proposed merge.
* **Apply merges** in batches, then re‑derive flags so each group has a single canonical master and every member points to it.
* **Publish Gold**: a deduped “masters‑only” list and a member‑to‑entity mapping that downstream apps can trust.
* **Carry decisions forward**: past merges feed back into the next day’s run so we don’t re‑score or re‑ask.

**Key personas**

* **Data/ML engineer**: operates notebooks, monitors runs, maintains configs and thresholds.
* **Reviewer**: resolves ambiguous pairs in the review app (business expert).
* **Admin**: can approve/reject in the Admin Gate for policy or compliance reasons.
* **Downstream consumer**: reads the Gold tables (analytics, MDM, CRM syncs).

**Where human input happens**

1. **LLM gray band** → automatic judgment; if **NEEDS\_CONFIRMATION** or low confidence, the pair is queued for **reviewers**.
2. **Admin Gate** → optional final supervision layer; admins can **APPROVE/REJECT** any pair before merges are applied.
3. **Reviewer notes** persist and are injected into future LLM prompts; re‑runs can be scoped to **FOCAL** or **COMPONENT** (connected graph) to minimize re‑work.

**Invariants & safety rails**

* Tables are **append‑only** ledgers (where practical). We compute “latest” snapshots with windowing.
* Already‑finalized YES/NO decisions are **anti‑joined** from new similarity work.
* Historical merges are **re‑applied first** (CDC) before computing new similarities.
* Merge application is **idempotent**: we recompute group flags from first principles (never flip flags by hand).
* All merges are **reversible**: the full proposal + decision history is retained.

**Typical outcomes by route**

* **Perfect match** → grouped immediately; flags set in the base table.
* **AUTO ≥ 0.95** → included in proposals without review (still visible in preview).
* **LLM YES** → included in proposals with LLM confidence & reason.
* **LLM NEEDS\_CONFIRMATION** → enters review queue; can be re‑asked with notes; YES after notes can auto‑approve if confidence ≥ policy.
* **LLM/auto NO** → simply not proposed; remains separate masters.

**CDC (Change Data Capture) behavior**

* We maintain a cumulative mapping **old\_master\_id → canonical\_master\_id**. Each Notebook 1 run applies it **before** building similarities (“do not re‑discover solved work”).
* Notebook 2 appends only the **delta** mapping produced by new merges.
* On the next run, the **latest‑wins** cumulative map is applied again, so results remain consistent and incremental.

# 3. High‑Level Architecture & Data Flow

Lakehouse layers are used: Bronze (raw), Silver (dedupe logic & proposals), Gold (final entities & lineage).

### High‑level data flow diagram

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 1: Transformations Notebook

A diagram of a computer program

AI-generated content may be incorrect.

Figure 2: Merges Notebooks

## Mermaid flowchart (copy into Confluence)

**Flowchart**

```mermaid  
flowchart LR

  subgraph Bronze

    B1[accounts\_raw\nraw accounts]

  end

  subgraph Silver

    S1[accounts\_full\_match.silver\nPerfect match + flags]

    S1b[CDC remap\nclean\_proposals\_accumulated.silver]

    S2[accounts\_full\_match\_filter.silver\nmasters only]

    S3[similarity\_pairs.silver\npairwise scores]

    S4[similarity\_candidates.silver\nroute=AUTO\_95 / LLM / NO]

    S5[llm\_jobs.silver\ntop-N per focal, batched]

    S6[llm\_results.silver\narray decisions]

    S7[review\_queue.silver\nNEEDS / guardrail]

    S8[apply\_proposals.silver\nAUTO\_95 + LLM\_YES + HUMAN]

    S9[clean\_proposals.silver\ncomponent collapse → mapping]

    SH[decisions\_history.silver\nYES / NO ledger]

    SA[clean\_proposals\_accumulated.silver\nCDC carry-forward]

  end

  subgraph Gold

    G1[accounts\_full\_match\_filter.gold\nfinal masters]

    G2[entity\_members]

    G3[merge\_audit]

  end

  B1 --> S1 --> S1b --> S2 --> S3 --> S4

  S4 -- LLM --> S5 --> S6 --> S7

  S4 -- AUTO\_95 --> S8

  S6 -- "LLM YES **(**high conf**)**" --> S8

  S7 -- "human approved" --> S8

  S8 --> S9 --> G1

  G1 --> G2

  S9 --> G3

  S9 --> SA

  SH -. anti-join .- S3  
```

# 4. Tables & Paths

## 4.1 Fabric table names (canonical)

**How to build each table (exact logic)**

**Conventions used below**

* Thresholds: T\_AUTO = 0.95, T\_LLM\_LOW = 0.70, TOP\_N = k.
* pair\_key = concat\_ws('|', least(master\_a\_id, master\_b\_id), greatest(master\_a\_id, master\_b\_id))
* “append\_per\_run, partition: run\_id” = write a new partition every run (CDC-friendly snapshots).

**1) accounts\_raw (Bronze)**

* **Purpose**: raw ingest.
* **Write**: append\_per\_run, partition by run\_id.
* **Columns**: account\_id, account\_name, [normalized\_name optional], run\_id.
* **Logic**: simply land the source rows; if normalized\_name isn’t provided, compute it downstream.

**2) accounts\_full\_match.silver (base + perfect match + CDC applied)**

* **Inputs**: accounts\_raw, clean\_proposals\_accumulated.silver (CDC map from prior runs).
* **Write**: append\_per\_run, partition run\_id.
* **Columns**:  
  account\_id, account\_name, normalized\_name, master\_account\_id, group\_size, is\_master, is\_dupe, stage\_rule, run\_id.
* **Steps**:
  1. normalized\_name = normalize(account\_name) (lowercase, strip punctuation, collapse whitespace).
  2. master\_account\_id = MIN(account\_id) BY normalized\_name (pick a stable keeper per exact normalized name).
  3. group\_size = COUNT(\*) BY master\_account\_id.
  4. Flags:
     + is\_master = account\_id = master\_account\_id
     + is\_dupe = group\_size > 1
     + stage\_rule = 'PERFECT' WHERE is\_dupe.
  5. **CDC** (if any mapping exists already): join on master\_account\_id = old\_master\_id and rewrite to canonical\_master\_id. (If you prefer, apply CDC at the top before steps 2–4.)

**3) accounts\_full\_match\_filter.silver (masters slice)**

* **Input**: accounts\_full\_match.silver.
* **Write**: append\_per\_run, partition run\_id.
* **Columns**: account\_id (== master), account\_name, master\_account\_id, group\_size, is\_master, is\_dupe, run\_id.
* **Filter**: account\_id = master\_account\_id.  
  This is the masters-only view used to compute similarities.

**4) similarity\_pairs.silver (all master A<B pairs with similarity score)**

* **Input**: accounts\_full\_match\_filter.silver.
* **Write**: append\_per\_run, partition run\_id.
* **Columns**: pair\_key, master\_a\_id, master\_b\_id, name\_a, name\_b, score, run\_id.
* **Steps**:
  1. Compute embedding = embeddings(normalized\_name) (Azure OpenAI text-embedding deployment).
  2. Form all unique master pairs with **A<B** (no self-pairs).
  3. score = cosine(embedding[A], embedding[B]).
  4. pair\_key = concat\_ws('|', least(A,B), greatest(A,B)).

**5) similarity\_candidates.silver (routing labels)**

* **Inputs**: similarity\_pairs.silver, decisions\_history.silver (for anti-join YES).
* **Write**: append\_per\_run, partition run\_id.
* **Columns**:  
  pair\_key, master\_a\_id, master\_b\_id, score, route, proposed\_keep\_master, proposed\_merge\_master, policy, threshold\_version, tagged\_at, run\_id.
* **Steps**:
  1. **Anti-join** pairs that already have final\_decision='YES' in decisions\_history (skip rework).
  2. Route:
     + AUTO\_YES\_95 if score ≥ T\_AUTO
     + LLM if T\_LLM\_LOW ≤ score < T\_AUTO
     + AUTO\_NO otherwise.
  3. For AUTO\_YES\_95: set proposed\_keep\_master = master\_a\_id, proposed\_merge\_master = master\_b\_id.
  4. Add tags: policy='first\_wins', threshold\_version='v1', tagged\_at = now().

**6) llm\_jobs.silver (top-N candidates per focal in LLM band)**

* **Input**: similarity\_candidates.silver.
* **Write**: append\_per\_run, partition run\_id.
* **Columns**: focal\_master\_id, candidates:[{candidate\_master\_id, score, rank}], model\_name, prompt\_version, created\_at, run\_id.
* **Steps**:
  1. Filter route='LLM'.
  2. Window partition by master\_a\_id, order by score DESC; keep top-N.
  3. collect\_list into candidates[].
  4. Add model\_name (e.g., gpt-4.1), prompt\_version, created\_at.

**7) llm\_results.silver (structured LLM outcomes)**

* **Inputs**: llm\_jobs.silver, plus candidate & focal names from accounts\_full\_match\_filter.silver for better prompts.
* **Write**: append\_per\_run, partition run\_id.
* **Columns**:  
  focal\_master\_id, focal\_name, results:[{candidate\_master\_id, candidate\_name, score, llm\_decision, llm\_confidence, llm\_reason}], model\_name, prompt\_version, decided\_at, run\_id.
* **Steps**:
  1. Call Azure OpenAI with tool-calling. The tool result must emit **one record per focal** with the results[] array containing YES / NO / NEEDS\_CONFIRMATION, with confidences and reason strings.
  2. Store raw outcomes + metadata.

**8) review\_queue.silver (human review snapshot)**

* **Input**: llm\_results.silver (explode results[]).
* **Write**: **snapshot\_overwrite** (latest state per run), not append.
* **Columns**:  
  pair\_key, focal\_master\_id, candidate\_master\_id, score, llm\_decision, llm\_confidence, llm\_reason, queue\_reason, status, reviewer, notes, enqueued\_at, updated\_at.
* **Rule to queue** (any true):
  + llm\_decision == 'NEEDS\_CONFIRMATION'
  + llm\_decision == 'YES' and llm\_confidence < 0.70
  + llm\_decision == 'NO' and llm\_confidence < 0.60 and score near T\_LLM\_LOW
* **Notes**: your app writes status to APPROVED / REJECTED and may attach notes; notebook re-reads it next run.

**9) apply\_proposals.silver (raw YES ledger for this run)**

* **Inputs**:  
  similarity\_candidates.silver (AUTO\_YES\_95),  
  llm\_results.silver (YES),  
  review\_queue.silver (status='APPROVED').
* **Write**: append\_per\_run, partition run\_id.
* **Columns**: pair\_key, m1, m2, source, score, decided\_at, run\_id.
* **Steps**:
  1. From AUTO\_YES\_95: source='AUTO\_95'.
  2. From LLM: keep only llm\_decision='YES', source='LLM\_YES'.
  3. From review queue: status='APPROVED', source='HUMAN\_APPROVED'.
  4. (Optional) Remap m1/m2 to **current canonical ids** using the accumulated map to avoid stale ids.

**10) admin\_gate.silver (optional)**

* **Purpose**: operators can APPROVE or REJECT pair\_keys outside the model/queue.
* **Write**: append (latest updated\_at wins per pair\_key).
* **Used**: to compute preview in the next step.

**11) final\_proposals.preview**

* **Inputs**: apply\_proposals.silver, admin\_gate.silver.
* **Write**: **snapshot\_overwrite** per run.
* **Logic**: AUTO\_95 ∪ LLM\_YES ∪ HUMAN\_APPROVED ∪ ADMIN\_APPROVED – ADMIN\_REJECTS.

**12) clean\_proposals.silver (de-dup & canonical keeper)**

* **Input**: final\_proposals.preview, decisions\_history.silver, clean\_proposals\_accumulated.silver (to refresh ids).
* **Write**: snapshot\_overwrite.
* **Columns**: old\_master\_id, canonical\_master\_id, source, score, decided\_at, run\_id.
* **Steps**:
  1. Drop duplicate (keep, merge) edges.
  2. **Safety anti-join**: exclude pair\_keys that already have final\_decision='YES' in decisions\_history.
  3. Refresh any stale masters using accumulated map (map both endpoints to current canonical).
  4. Build undirected graph of proposed merges; find connected components.
  5. Pick one canonical keeper per component (e.g., lowest id).
  6. Emit **mapping rows** for **every non-keeper member**: old\_master\_id → canonical\_master\_id.

**13) clean\_proposals\_accumulated.silver (historical CDC map)**

* **Input**: clean\_proposals.silver.
* **Write**: **snapshot\_overwrite** (latest wins by old\_master\_id).
* **Use**: this is the **CDC map** fed back into Notebook 1 (step 2) before matching/similarity next run.

**14) accounts\_full\_match.silver.postapply (base rewritten with mapping)**

* **Inputs**: accounts\_full\_match.silver, clean\_proposals\_accumulated.silver.
* **Write**: append\_per\_run, partition run\_id.
* **Logic**:
  1. Left join on master\_account\_id = old\_master\_id; rewrite to canonical\_master\_id if present.
  2. Recompute group\_size by master\_account\_id.
  3. Refresh flags:
     + is\_master = account\_id = master\_account\_id
     + is\_dupe = group\_size > 1.

**15) accounts\_full\_match\_filter.gold (masters-only deduped list)**

* **Input**: accounts\_full\_match.silver.postapply.
* **Write**: append\_per\_run, partition run\_id.
* **Filter**: account\_id = master\_account\_id.
* **This is your published deduped list (one row per canonical entity).**

*(Optionally you can also materialize entities\_gold and entity\_members if you prefer an explicit entities/members model, but the above Gold already serves as “one row per current master”.)*

**16) decisions\_history.silver (finalization ledger)**

* **Purpose**: immutable audit for downstream anti-join.
* **Write**: append.
* **On finalize** of a run:
  + Append YES from final\_proposals.preview (and the explicit admin NO vetoes).
  + Upstream step 5 anti-joins using **YES** only (so already-merged pairs never reappear).

## 4.2 Local CLI artifact paths (from code)

* ROOT: ./output/azure\_cicd\_cli/
* Decisions ledger CSV: ./output/azure\_cicd\_cli/decisions\_history.csv
* Cumulative mapping CSV: ./output/azure\_cicd\_cli/clean\_proposals\_accumulated.csv
* Per‑run snapshots under: ./output/azure\_cicd\_cli/cycle\_YYYYMMDD\_HHMMSS/

# 5. Stage‑by‑Stage Implementation

## Stage 1 – Perfect match

Normalize names, group exact matches, assign master\_account\_id, derive flags (is\_master, is\_dupe, group\_size) into accounts\_full\_match.silver.

## Stage 1b – CDC remap

Apply clean\_proposals\_accumulated.silver to refresh master pointers before similarities; re-derive flags.

## Stage 2 – Masters slice

accounts\_full\_match\_filter.silver = is\_master = true (masters only).

## Stage 3 – Similarity

Compute embeddings + cosine; optional fuzzy blend (weights SIM\_EMB\_WEIGHT/SIM\_FUZZ\_WEIGHT). Emit canonical pairs (A|B) into similarity\_pairs.silver; anti-join decisions\_history.silver.

## Stage 4 – Routing

similarity\_candidates.silver tags route=AUTO\_YES\_95 (≥T\_AUTO), LLM (≥T\_LLM\_LOW & <T\_AUTO), AUTO\_NO (<T\_LLM\_LOW). For AUTO, set proposed\_keep\_master=master\_a\_id.

## Stage 5 – LLM jobs

Group LLM-band by focal master, keep top-N by score (LLM\_TOP\_N). Chunk per request by LLM\_ARRAY\_SIZE. Persist in llm\_jobs.silver.

## Stage 6 – LLM results

Call AOAI; record per focal array decisions with {candidate\_id, score, llm\_decision, llm\_confidence, llm\_reason, context\_used} into llm\_results.silver.

## Stage 7 – Review queue

Explode results; enqueue NEEDS\_CONFIRMATION and guardrail cases into review\_queue.silver with status=QUEUED.

## Stage 8 – Final validation (optional admin)

Confirm/override AUTO\_95 + LLM\_YES; append YES/NO to decisions\_history.silver; build apply\_proposals.silver.

## Stage 9 – Clean mapping

Collapse components (union‑find) to produce conflict‑free clean\_proposals.silver (old\_master\_id → canonical\_master\_id). Append to clean\_proposals\_accumulated.silver.

## Stage 10 – Apply mapping

Rewrite master pointers in accounts\_full\_match.silver; re‑derive flags; write merge\_audit.

## Stage 11 – Publish Gold

Produce accounts\_full\_match\_filter.gold (final masters) and entity\_members.

# 6. Reviewer Workflows & Triggers

## 6.1 Interactive human review (queue)

* Review items in review\_queue.silver where status=QUEUED.
* Actions: Approve (YES), Reject (NO), or Skip; optionally add reviewer notes.
* Optional LLM re‑run with notes; auto‑approve flips to YES above AUTO\_APPROVE\_RERUN\_YES\_CONF (except current pair).

Rerun scope controls (all | foci | foci\_related) ensure notes propagate to the right subset without over‑calling the LLM.

**NOTE:** Reviewer notes persist in the Context Book (global + per‑focal) and are automatically appended to future LLM prompts—including daily Notebook 1 runs—so improvements scale forward with no extra action.

## 6.2 Bulk re‑run with notes

* Use rerun scope to rejudge ALL / FOCI / FOCI+RELATED queued pairs with accumulated notes.
* Confident YES decisions can auto‑approve per threshold; queue is refreshed accordingly.

# 7. Configuration & Environment

This section captures **all runtime parameters** and environment variables used by both notebooks, with sensible defaults, purpose, and safety notes. Use these to tune behavior per environment (Dev → UAT → Prod) without modifying code.

**Notation**  
Notebook 1 = Transform (CDC → Perfect → Similarity → Routing → LLM/Queue)  
Notebook 2 = Merge (Proposals → Admin Gate → Clean → Apply → Gold)

## 7.1 Core thresholds & routing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Default | Used In | Purpose | Notes |
| T\_AUTO | 0.95 | NB1 | Auto‑merge threshold (routes pair to AUTO\_YES\_95). | Raise to reduce false merges; lower to be more aggressive. |
| T\_LLM\_LOW | 0.70 | NB1 | Lower bound of LLM band. Pairs in [T\_LLM\_LOW, T\_AUTO) go to LLM. | Keep a safe gap from T\_AUTO (e.g., 0.20–0.30) to reduce churn. |
| LLM\_TOP\_N | 3 | NB1 | Top‑N candidates sent to LLM per focal (ranked by score). | Drives LLM workload. |
| LLM\_BATCH\_SIZE | LLM\_TOP\_N | NB1 | Max candidates per LLM call; larger values reduce API round‑trips. | Requests are chunked per focal into batches of LLM\_BATCH\_SIZE. |

## 7.2 Rerun scope & controls (Notebook 1)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Default | Purpose | Example |
| RERUN\_SCOPE | all | Restrict LLM reruns: all | focal | component. | component reruns only the graph‑connected focals to selected seeds. |
| RERUN\_FOCALS | "" | Comma‑separated focal IDs (only for focal/component). | a1,a3,g3 |
| RERUN\_PAIR\_KEYS | "" | Comma‑separated pair\_keys to force rerun. | `a1 |
| RERUN\_ALL\_LLM\_QUEUED | false | If true, rerun **all** queued pairs regardless of seeds. | Useful after new global guidance. |
| RERUN\_NOTES\_JSON | "" | JSON map {pair\_key: note} injected into LLM context. | `{ "a1 |
| FOCAL\_NOTES\_JSON | "" | JSON map {focal\_id: note} injected for all that focal's pairs. | { "g3": "Intl ~ International" } |
| GLOBAL\_NOTES | "" | A global note appended to every LLM rerun prompt. | "Ignore punctuation variants of LLC" |
| AUTO\_APPROVE\_RERUN\_YES | true | If rerun produces YES with high confidence, auto‑approve that queued pair. | Works with AUTO\_APPROVE\_RERUN\_YES\_CONF. |
| AUTO\_APPROVE\_RERUN\_YES\_CONF | 0.9 | Confidence threshold to auto‑approve a rerun YES. | e.g., 0.92 in stricter prod. |

## 7.3 Admin policies (Notebook 2)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Default | Purpose | Notes |
| FINALIZE | false | When true, writes to decisions\_history.silver and clean\_proposals\_accumulated.silver, then re‑applies mapping to produce PostApply + Gold. | In UAT, keep false to review final\_proposals.preview before committing. |
| ADMIN\_REVIEW | false | If true, require admin confirmation for **all** suggested merges (ignore auto + LLM YES until admin approves). | Can be used temporarily during go‑live. |

**How ADMIN\_REVIEW works:** When enabled, downstream pipelines must treat final\_proposals.preview as advisory. Only rows that also appear with APPROVE in admin\_gate.silver should be treated as effective proposals.

## 7.4 Similarity composition (optional)

If you introduce hybrid similarity (embeddings + fuzzy/Levenshtein), expose the weights as env vars and compute a combined score used for routing thresholds.

|  |  |  |
| --- | --- | --- |
| Name | Default | Purpose |
| SIM\_EMB\_WEIGHT | 0.95 | Weight for embedding similarity in combined score. |
| SIM\_FUZZ\_WEIGHT | 0.05 | Weight for fuzzy similarity in combined score. |

**Note:** In the current notebooks we use embeddings only for score. If you add fuzzy components, ensure the combined score still maps to calibrated thresholds (T\_AUTO, T\_LLM\_LOW).

## 7.5 Azure OpenAI (AOAI)

|  |  |  |
| --- | --- | --- |
| Name | Default | Purpose |
| AOAI\_ENDPOINT | — | Azure OpenAI endpoint (e.g., https://<resource>.openai.azure.com). |
| AOAI\_CHAT\_DEPLOYMENT | — | Deployed chat model name (e.g., gpt-4.1). |
| AOAI\_EMBEDDING\_DEPLOYMENT | — | Embedding deployment name (e.g., text-embedding-3-small). |
| AOAI\_API\_VERSION | 2024-08-01-preview | API version. |

**Auth:** Both notebooks use **Managed Identity** via DefaultAzureCredential. Ensure the workspace identity has Cognitive Services OpenAI User on the AOAI resource.

## 7.6 Paths & tables

Set a single base directory per environment and the notebooks will write/read all tables beneath it.

|  |  |  |
| --- | --- | --- |
| Name | Default | Example |
| BASE\_DIR | /lake/dedupe\_demo | /lake/prod/dedupe\_fabric |

All tables are Parquet under ${BASE\_DIR}/<table\_name> and are append-only unless otherwise noted in your doc’s table list. run\_id is used to partition per-run snapshots.

## 7.7 Operational guidance

* **Idempotence:** Rerunning Notebook 1 with the same RUN\_ID should be avoided; prefer new RUN\_IDs. Notebook 2 reads by RUN\_ID for preview and commits only when FINALIZE=true.
* **Backpressure:** If queue grows, set RERUN\_SCOPE="component" with focal seeds to minimize LLM calls while still resolving connected duplicates.
* **Observability:** Persist aoai\_request\_id, token\_usage\_json, prompt\_version, context\_hash in LLM results for auditability.
* **Calibration:** Start with conservative T\_AUTO=0.97, then lower gradually as you validate precision.
* **Disaster recovery:** Because tables are append-only, you can reconstruct any run by filtering by run\_id and ignoring later deltas.

# 8. Caveats, Particularities & Safety Guards

* Pair canonicalization: store pairs as A|B (sorted) and drop self‑pairs.
* Anti‑replay: anti‑join decisions\_history.silver for pairs and jobs so finalized YES/NO never reappear.
* Component collapse: merges are order‑independent; one canonical keeper per connected component (lowest ID policy).
* Idempotency: re‑derive flags from master pointers; never hand‑flip booleans.
* Never‑merge: add explicit NO to decisions\_history.silver to permanently suppress suggestions.
* Rerun scope: all | foci | foci\_related controls coverage and cost of LLM rejudging.
* Cost controls: top‑N per focal + array batching; use seeds to target high‑value clusters.
* Security: AOAI uses Azure AD token provider; no secrets in code; ensure least‑privileged access to tables.

# 9. Operational Runbook

## 9.1 Schedules & Roles

* Daily: Notebook 1 (end‑to‑end detect) – produces pairs, routing, LLM jobs/results, and the review queue.
* Weekly or on‑demand: Notebook 2 (final validation + apply) – consolidates approvals, writes ledgers, applies merges, publishes Gold.
* Roles: Data Stewards (review queue), Admin (optional final validation), Engineering (maintenance & thresholds), Platform Ops (scheduling, access).

## 9.2 SLAs & Quality KPIs

* Precision@Auto≥95, LLM acceptance rate, Queue backlog, Time‑to‑decision, Merge volume by source (AUTO/LLM/HUMAN).
* Drift monitors: score distributions, LLM confidence distribution, top reasons, repeated NOs.

## 9.3 Troubleshooting

* ‘Wrong auto‑merge’: add NO to decisions\_history.silver and rerun—order‑independent component collapse will correct next run.
* ‘Missed duplicate’: lower thresholds or add CDC mapping to clean\_proposals\_accumulated.silver and rerun.
* ‘Same pair keeps resurfacing’: ensure NO is recorded in the ledger; confirm pair\_key is canonical (A|B).
* ‘LLM variance’: increase LLM\_ARRAY\_SIZE for more joint context or add reviewer notes; notes persist into future runs.

# 10. Diagrams — Sequence (copy into Confluence)

**Daily detect + periodic apply**

```mermaid  
sequenceDiagram  
 participant Scheduler as Daily Scheduler  
 participant NB1 as Notebook 1 (Detect)  
 participant LLM as Azure OpenAI  
 participant Steward as Data Steward  
 participant NB2 as Notebook 2 (Apply)  
  
 Scheduler->>NB1: Run daily  
 NB1->>NB1: Perfect match + CDC remap  
 NB1->>NB1: Similarity → Routing  
 NB1->>LLM: Batch LLM calls for grey band  
 LLM-->>NB1: Decisions (YES/NO/NEEDS + confidence + context\_used)  
 NB1->>NB1: Build review\_queue (NEEDS / guardrails)  
 NB1-->>Steward: Queue ready for review  
 Steward->>NB1: Approve/Reject + Notes (persist)  
 Steward->>NB1: Optional rerun (scope all/foci/foci\_related)  
 NB1->>LLM: Rejudge targeted queue with notes  
 LLM-->>NB1: Updated judgments (auto-approve flips per policy)  
 Scheduler-->>NB2: Weekly/on‑demand  
 NB2->>NB2: Consolidate AUTO\_95 + LLM\_YES + HUMAN  
 NB2->>NB2: Clean components → mapping  
 NB2->>NB2: Apply mapping, re‑derive flags  
 NB2-->>Steward: Publish Gold & audit  
```

**Automatic propagation of reviewer notes to future runs**

```mermaid  
sequenceDiagram  
 participant Steward  
 participant NB1 as Notebook 1  
 participant LLM as Azure OpenAI  
  
 Steward->>NB1: Add reviewer notes (global or per‑focal)  
 NB1->>LLM: Subsequent daily run uses accumulated notes automatically  
 LLM-->>NB1: Decisions reflect notes (context\_used echoed)  
 NB1-->>Steward: Reduced queue / higher confidence outcomes  
```

**Interaction Flow**

```mermaid  
flowchart TD

U[User / Streamlit UI] -->|Edit/Upload| A[accounts.bronze]

U -->|Start Deduplication| S1

subgraph Transform Notebook

S1[Perfect match base] --> S2[Apply cumulative mapping]

S2 --> S3[Masters slice]

S3 --> S4[Embeddings AOAI]

S4 --> S5[All pairwise similarities]

S5 --> S6[Routing\nAUTO\_95 / LLM / AUTO\_NO]

S6 -->|AUTO\_95| P1[apply\_proposals.silver auto]

S6 -->|LLM band| L1[llm\_jobs.silver]

L1 -->|batch, scope, seeds| L2[Call Azure OpenAI]

L2 --> L3[llm\_results.silver\nresults + context\_used]

L3 --> Q[review\_queue.silver\nNEEDS\_CONFIRMATION only]

end

subgraph Review App

RN[reviewer\_notes.silver] -.-> L1

Q --> T1[LLM Review table NEEDS\n approve/reject + notes + include\_in\_rerun]

T1 -->|Save notes| RN

T1 -->|Apply Notes & Rerun all/foci/foci\_related| L2

T1 -->|Persist approvals buffer| H{human\_decisions in UI}

S6 --> T2[Final table ALL pairs\nAUTO + LLM + HUMAN]

L3 --> T2

H --> T2

T2 -->|Preview union-find impact| PV[Temporary clean\_proposals → preview masters/full]

T2 -->|Execute merges| EX[apply\_proposals.silver\nAUTO\_95 + LLM\_YES + HUMAN/ADMIN]

end

EX --> CP[clean\_proposals.silver\nunion-find + component root policy]

CP --> MAP[clean\_proposals\_accumulated.silver\ncompressed]

MAP --> R1[Re-apply mapping]

R1 --> GOLD[accounts\_full\_match\_filter.gold masters\_gold]

GOLD --> U

U -->|Start next cycle| S1  
```

# 11. Change Management

* Thresholds/policies live in environment or config sheet; changes are versioned in run metadata.
* Normalization updates trigger recompute from Stage 1; document changes with examples.
* Model/version changes are captured in llm\_results.silver (model\_name, prompt\_version).

# 12. Glossary

* Master: keeper row representing an entity; others map to it.
* Entity: the canonical real‑world account (equals master in this design).
* Route: AUTO\_YES\_95 / LLM / AUTO\_NO triage bucket.
* Component: connected set of masters that collapse to a single keeper.
* CDC: carry‑forward of prior merges via accumulated mapping.

# Addendum — Worked Example (T0 & T+1)

This addendum clarifies **what is sent at each stage** (payload shapes, where records flow), and **how the system behaves when new accounts arrive** while some pairs are still in the review queue.

**Notation**  
**NB1** = Transform notebook (CDC → Perfect → Similarity → Routing → LLM/Queue)  
**NB2** = Merge notebook (Proposals → Admin Gate → Clean → Apply → Gold)  
pair\_key = concat\_ws("|", least(A,B), greatest(A,B))  
Tables are append‑only unless otherwise noted and partitioned by run\_id.

**1) Confirmation — Scope When New Accounts Arrive**

**Default/Recommended behavior:**

* When a new batch arrives (e.g., +100 accounts), **NB1 recomputes on *ALL current masters*** **after CDC** (i.e., after re‑applying the cumulative mapping from prior finalized decisions).
* **Queued pairs are *not* merged yet**, so their two masters remain distinct and are included in pairing against the new accounts.
* We **anti‑join only pairs already decided YES** in decisions\_history.silver. Queued pairs are still eligible to be reconsidered/rerun (e.g., if reviewer notes were added).

**Why not restrict to only "accepted masters + queued focals"?**

* New accounts may be similar to masters that aren’t currently queued. Limiting the pairing scope risks **missed matches** and fragmentation.
* If cost is a concern, you can run **RERUN\_SCOPE=component** for targeted reruns on queued subsets, but daily/regular NB1 should still cover **all masters** to maintain global dedupe quality.

**In short:** Recompute similarities for **all masters after CDC**; skip already‑finalized YES pairs; preserve/reuse the review queue for unresolved pairs.

**2) T0 Example — What Each Step *Sends***

**Stage A — Perfect Match & CDC (NB1)**

**Inputs:** accounts\_raw (this run).  
**CDC Apply:** clean\_proposals\_accumulated.silver (latest‑wins map) is joined to rewrite master\_account\_id before any similarity work.  
**Outputs:**

* accounts\_full\_match.silver *(append per run)* — Snapshot with flags after perfect match + CDC.
* accounts\_full\_match\_filter.silver *(append per run)* — **Masters slice** for similarity (one row per master in this run).

*Nothing is sent externally in this stage — these are internal tables for downstream steps.*

**Stage B — Embeddings & Pairing (NB1)**

**Embeddings request (to AOAI):** batched array of normalized master names.

{

"model": "${AOAI\_EMBEDDING\_DEPLOYMENT}",

"input": [

"acme corp", "acme int", "globex llc", "globex intl logistics", "zeta holdings", ...

]

}

**Embeddings response (per batch):** list of vectors (omitted here for brevity).

**Pair generation:** Cross‑join masters **A<B** with cosine on embeddings.  
**Anti‑join:** drop any pair\_key that already has decision=YES in decisions\_history.silver.

**Outputs:**

* similarity\_pairs.silver — All pairs A<B with score (for this run, after anti‑join).
* similarity\_candidates.silver — Adds route:
  + AUTO\_YES\_95 if score ≥ T\_AUTO
  + LLM if T\_LLM\_LOW ≤ score < T\_AUTO
  + AUTO\_NO otherwise

**Stage C — LLM Jobs & Results (NB1)**

**What gets sent to the LLM (per job, per focal):**

{

"focal": {"id": "g1", "name": "Globex LLC"},

"candidates": [

{"id": "g2", "name": "Globex, L.L.C.", "score": 0.91},

{"id": "g4", "name": "Globex Int Logistics", "score": 0.80}

],

"notes": "(optional) reviewer/focal/global guidance",

"prompt\_version": "v1"

}

**Expected tool/function call response:**

{

"decisions": [

{

"candidate\_master\_id": "g2",

"decision": "YES",

"confidence": 0.88,

"score": 0.91,

"reason": "LLC punctuation variant"

},

{

"candidate\_master\_id": "g4",

"decision": "NEEDS\_CONFIRMATION",

"confidence": 0.62,

"score": 0.80,

"reason": "Intl ~ International but context unclear"

}

]

}

**Outputs:**

* llm\_jobs.silver — Grouped top‑N candidate payloads per focal (append).
* llm\_results.silver — The LLM’s structured decisions (array of per‑candidate structs) with **prompt metadata** (append).

**Stage D — Review Queue (NB1)**

**Queue appends:** explode llm\_results.silver (this run) → append entries where:

* NEEDS\_CONFIRMATION, or
* borderline YES (low confidence), or
* borderline NO near the threshold.

**Output:**

* review\_queue.silver — **Append‑only log** of queue events (we never overwrite; consumers build "latest" via windowing).  
  Fields include: pair\_key, focal\_master\_id, candidate\_master\_id, score, llm\_confidence, llm\_reason, status, notes, timestamps, etc.

If a rerun happens in the same run (e.g., notes added), new queue **updates** are appended as new rows (status may flip to APPROVED).

**Stage E — Compile Proposals & Preview (NB2)**

**Inputs:**

* similarity\_candidates.silver → AUTO\_YES\_95
* llm\_results.silver → LLM\_YES
* review\_queue.silver → HUMAN\_APPROVED (and HUMAN\_NOTES\_AUTO\_APPROVED from NB1 auto‑approval logic)
* admin\_gate.silver (optional) → ADMIN\_APPROVED / REJECT

**Output:**

* final\_proposals.preview *(append per run)* — One row per pair\_key with source, score, confidence, reason, and metadata.
* If FINALIZE=false, nothing else is written.

**Stage F — Finalize (if FINALIZE=true, NB2)**

**Ledgers:**

* decisions\_history.silver *(append)* — YES from proposals; NO for admin rejects.
* Build **union‑find** over (m1,m2) to emit **mapping delta**:  
  clean\_proposals\_accumulated.silver *(append)* — rows old\_master\_id → canonical\_master\_id with timestamps.

**Post‑apply & Gold:**

* Re‑apply **effective** mapping (latest per old\_master\_id) to accounts\_full\_match.silver of this run:
  + accounts\_full\_match.silver.postapply *(append per run)*
  + accounts\_full\_match\_filter.gold *(append per run)* — **Final deduped masters list** for the run.

**3) T+1 Example — New 100 Accounts While Some Pairs Remain Queued**

Assume at **T0**:

* Some pairs finalized (YES); accumulated in clean\_proposals\_accumulated.silver.
* Some pairs are still QUEUED in review\_queue.silver.

At **T+1** (NB1):

1. **CDC First.** Load clean\_proposals\_accumulated.silver (latest‑wins) and **rewrite masters** before any similarity work.
2. **Masters Slice Rebuilt.** This contains **all masters after CDC**, including those in queued pairs (they’re still distinct masters until merged).
3. **Pairs Computed vs ALL Masters.** We cross‑join A<B across **all T+1 masters**; we **anti‑join only decided‑YES pairs** from decisions\_history.silver (so we do *not* recompute already‑merged pairs).
   * This means the queued masters are compared **against the new 100 accounts as well** — they can receive fresh candidates.
4. **Routing & LLM.** Same thresholds, fresh LLM jobs created only for LLM band, with top‑N per focal.
   * If reviewer notes were saved as env inputs (or injected via NB1 rerun), they are applied to the new LLM calls (context‑hash logged).
5. **Queue Append.** New NEEDS\_CONFIRMATION items are appended. Existing queued pairs remain until explicitly approved/rejected or auto‑approved by rerun rules.

At **T+1** (NB2):

* Build final\_proposals.preview from AUTO\_95, LLM\_YES, and any approvals (human/admin).
* If FINALIZE=true, append ledgers, emit mapping delta, and re‑apply to produce PostApply + Gold for **T+1**.

**Key point:** New arrivals are always compared against the **global set of masters after CDC**, ensuring enterprise‑wide consistency. Queued pairs do not block new matches; they simply remain unresolved until approved.

**4) Message/Record Sizes & Practical Tips**

* **LLM Jobs:** Controlled by LLM\_TOP\_N and LLM\_BATCH\_SIZE. For 100 new accounts, expect roughly O(N·K) candidates evaluated by LLM (N = new/affected masters; K = top‑N).
* **Queue Growth:** If the queue spikes, use **RERUN\_SCOPE=component** with seeds to address connected clusters without fanning out to the entire corpus.
* **Anti‑join coverage:** We purposely anti‑join only **YES** decisions. NO decisions (explicit admin rejects) are retained but do not block recomputation in later runs — this lets new context flip a prior NO if warranted.