DEDICATTED

dedicatted.com

AI Multi-Agent Systems for Data Modeling:

Workflows, Metrics, Impact

We've spent years building data pipelines, and one process never changes – moving from OLTP to OLAP. Even though transactional systems capture every event precisely, reshaping that into an analytical schema means remapping tables and re-running validations every time a new source or requirement shows up. It's the boring part nobody talks about, still it takes up to 40% of data engineers' time.

When LLM-powered multi-agent systems began appearing in production tooling, we didn't accept the hype headlines about "replacing" engineers. Instead, we asked a practical question: can AI really automate the repeatable pieces of data modeling and integration?

Over the past 2 years we embedded multi-agent setups into pre-prod and production pipelines and used them for focused workflows:

- Semantic extraction
- ERD generation
- Schema/field matching
- Test-case synthesis
- Drift detection

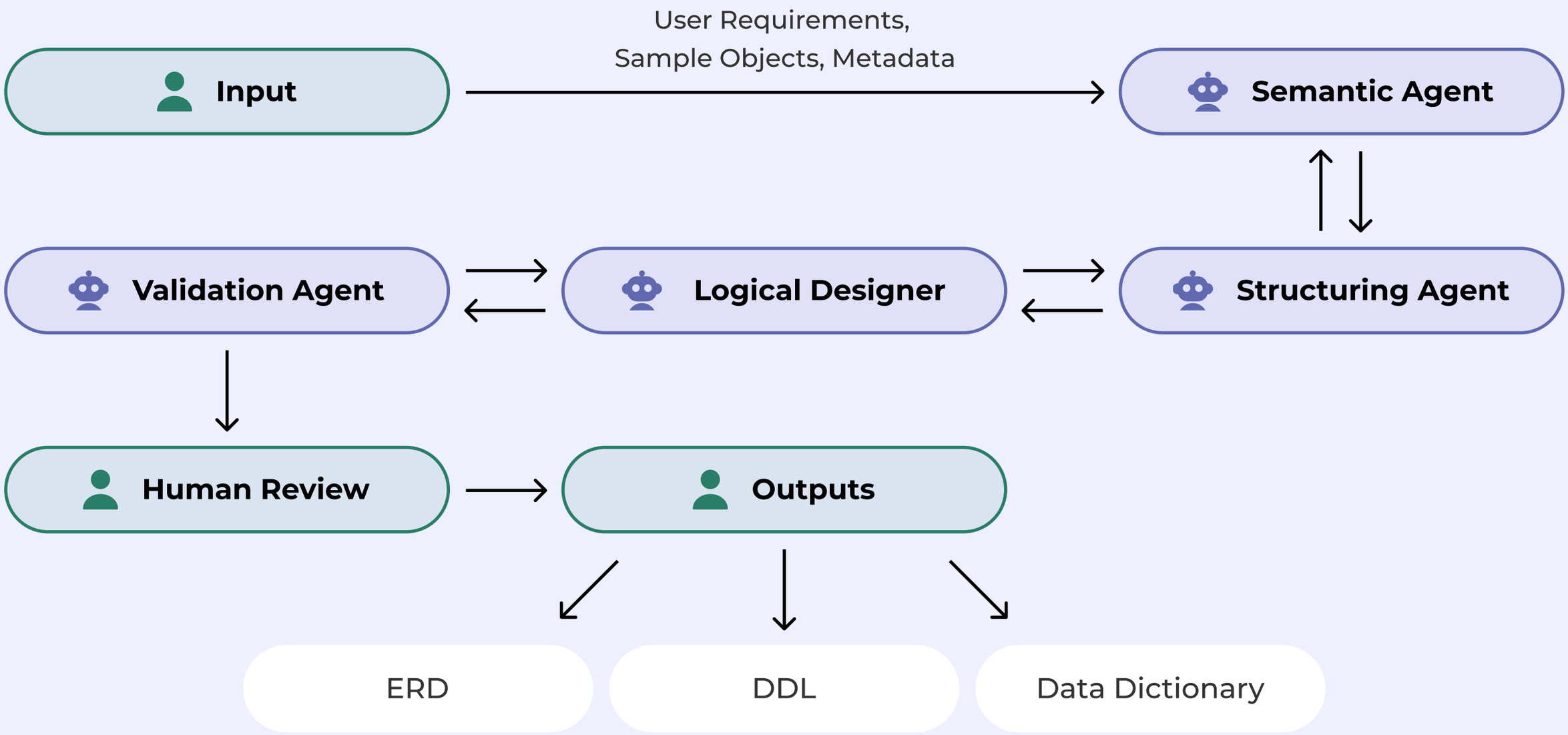
What we found from the pilots?

AI agents can produce concrete outputs: ERDs, DDLs, schema mappings with example rows and confidence scores, or fix suggestions with smoke tests, which **tend to shorten the entire path from the requirement to production-ready model/schema up to 3 times.**

This whitepaper is our attempt to make that experience actionable. We'll walk through the workflows we used, the metrics we tracked, and the design patterns that make agent outputs reviewable and auditable.

Automated Schema Generation

Here’s our first workflow, the one we employed on nearly every project: agents ingest requirement text, sample extracts and existing metadata, then run a sequence of prompts and verification steps to produce a conceptual ERD, a normalized logical model and executable DDL.



- **A semantic extraction agent** (LLM + prompt templates) - generates entity/attribute candidates.
- **A structuring agent** - normalizes relationships and proposes keys.
- **A logical-design agent** - transforms that ERD into normalized tables and a set of suggested keys and denormalizations.
- **A validation agent** - generates test queries and synthetic samples to validate the proposed model and report concrete failing cases.

In practice we started getting usable **first drafts in 5-7 days**, instead of weeks. Of course, that didn’t mean we could skip the human review step. E.g.: business rules (billing edge-cases, legacy flags, spreadsheet exceptions) tend to surface only when tests run or a domain expert looks at the results.

From our pilots

Pipelining schema fragments through AI multi-agents (extraction – modeling – validation) **produced first-draft schemas and DDL up to 5x faster than baseline, and reduced false positives by 78%.**

We also integrated some complementary features into the generation flow (the list below). They target the manual “detective” work that slows modelling and make agent outputs reviewable, testable and safe to apply.

- **Relationship inference**

  Agents analyze sample values, distinct counts, co-occurrence and naming patterns to propose candidate joins and foreign keys. Each proposed edge is shipped with example rows and a confidence score so reviewers can see why the join was suggested.

- **Validation & QA**

  The QA agent runs normalization & lossless-join checks, representative queries and synthetic edge-case tests. It emits failing test cases (with example rows) and suggested fixes. These concrete failing cases are what engineers review – not just warnings.

- **Documentation & Explainability**

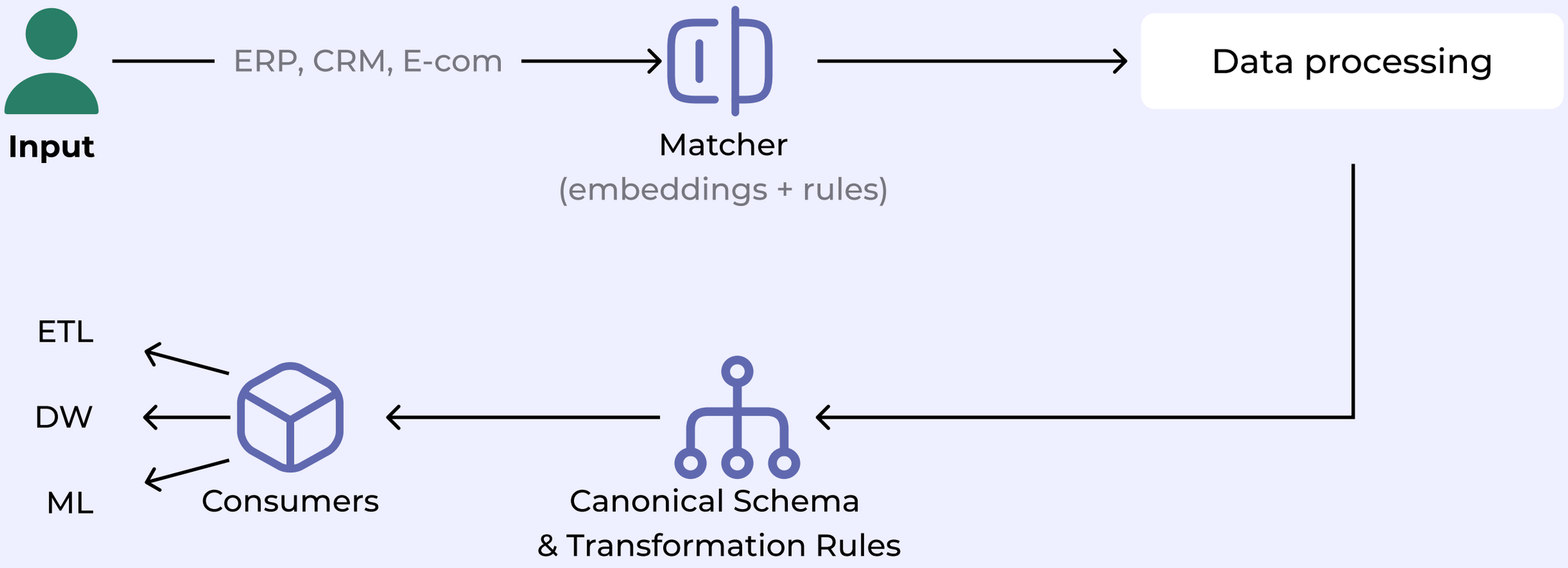
  Every inferred field is accompanied by a short data dictionary entry and a rationale (e.g., "matched on SKU pattern + 95% value overlap"). Deterministic outputs (ERD snapshot, DDL, data-dictionary entries) plus example rows make agent suggestions auditable and easy to import into catalogs.

- **Indexing & Performance Tuning**

  Usage-analytics agents analyze expected query shapes and cardinalities and run quick micro-benchmarks in a sandbox to recommend indexes, partitions or targeted denormalizations.

Schema Mapping & Integration

Integration is always about manual data work. Differing column names, encodings and small value quirks turn one-off ETL into debugging sessions. Below is a matcher we built to make mappings repeatable and easy to operationalize.



At a high level the matcher:

1. Computes vector similarity between columns (name + samples);
2. Compares cardinalities and value overlap;
3. Applies deterministic heuristics (date patterns, ID formats, codebooks).

For each candidate mapping it returns a short package: the suggested mapping, example row pairs, overlap percentage, a similarity score, and a suggested transformation snippet. Borderline mappings are surfaced in a review queue with this evidence so engineers can make quick decisions. Canonical schema outputs and the generated SQL/PySpark transforms are formatted so downstream ETL jobs can pick them up with minimal rework.

From our pilots

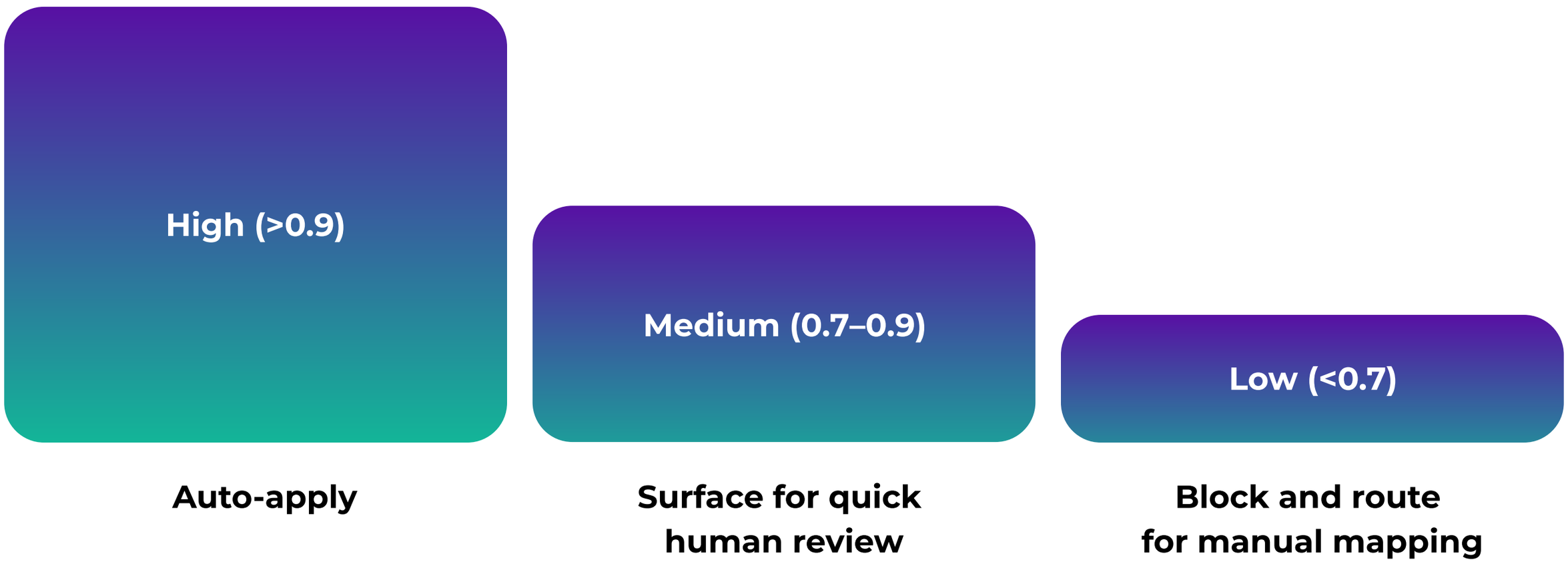
We've seen a hybrid matcher (embeddings + rules) move a large fraction of mappings into an "approve with evidence" state.

- ✓ **Auto-mapping ±20k assets with 98% precision in 9 minutes on a single deployment;**
- ✓ **Natural-language (NL) discovery interfaces speed domain lookups by up to ~100x in some workflows.**

Relationship inference also helps with canonicalization. When several sources contain related entities, agents propose consolidated entity definitions and candidate joins — each proposal comes with example rows and a confidence score. Those proposals are fed back into the canonical schema so the model gradually gets more complete and consistent.

We expose the canonical schema via a simple natural-language layer so business users can ask for fields or joins in plain English. The system returns the suggested mapping plus the evidence (example rows, similarity and overlap stats), so ambiguous translations are transparent and easy to review.

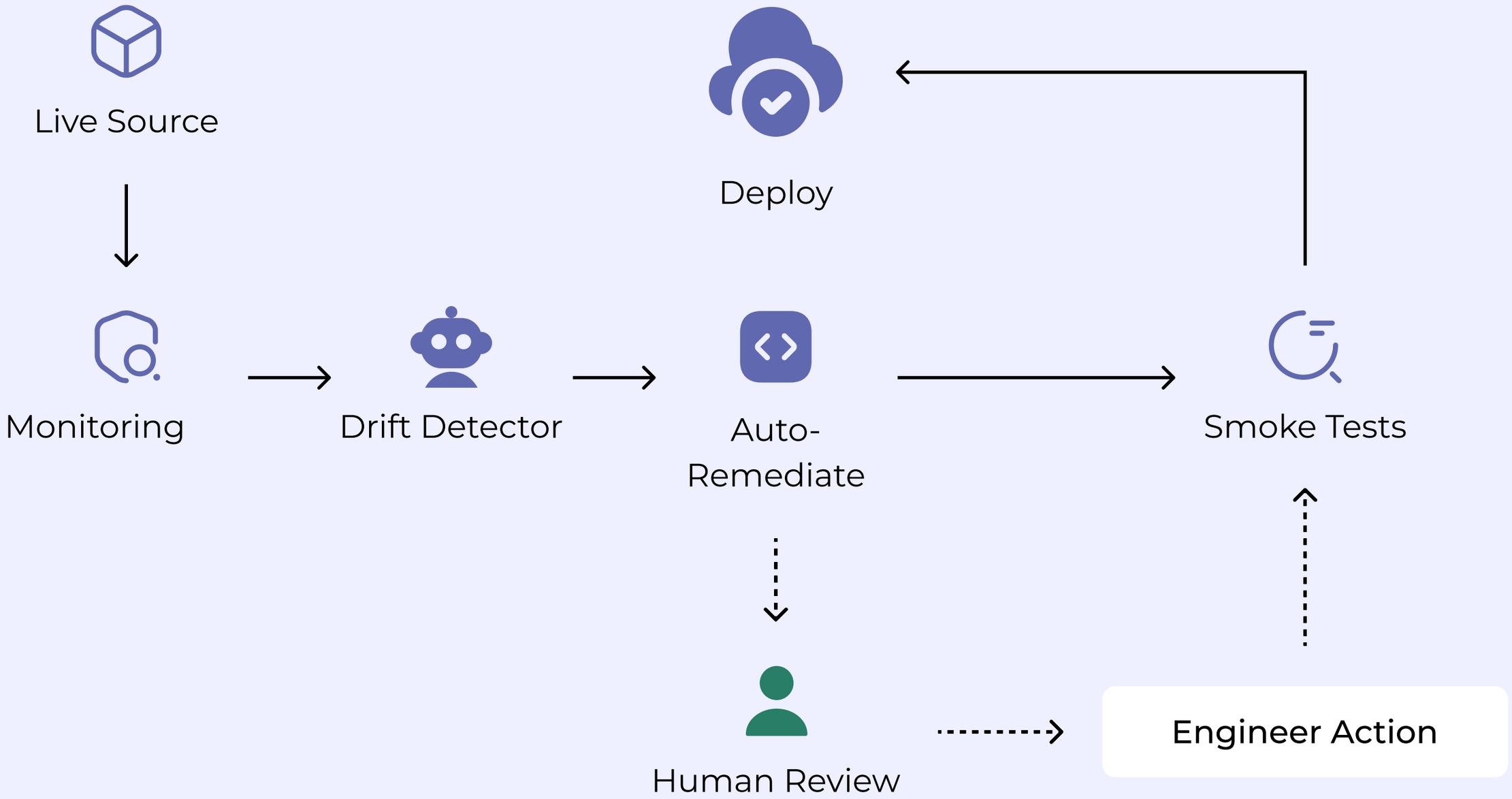
Here we gate changes per field by confidence:



All mappings, evidence snippets and rationales are recorded in the metadata catalog (with links to sample rows and any generated transform). That makes troubleshooting way easier.

Self-Healing Pipelines

Few things break pipelines as often as schema drift. It could be a renamed column, a new optional field, or a subtle type change can silently break dashboards, ETL jobs and models. Here we treat adaptation like a control loop: detect change, propose a fix, validate it, and then either apply or escalate.



Here’s how we run the loop:

1

Monitoring agents

Snapshot incoming schema shapes and basic stats — column lists and types, null rates, distinct counts, sample rows.

2

Detector agents

Compute schema diffs and drift metrics (distribution shifts, cardinality changes, new or removed columns).

3

Remediation agents

Propose concrete fixes (rename + cast, mapping entries, backfill queries) and attach a confidence score and a recommended action: auto-apply, deploy, or human review.

Though, schema drift fixes can be risky. We've learned the hard way that "just auto-fix everything" creates more incidents than it prevents. So, we suggest to keep this basic set of safety rules:

- **Keep auto-apply conservative**

  Only high-confidence fixes (clear rename with matching distributions, identical cardinality, or dictionary match).

- **Any auto-applied change must pass a smoke-test suite**

  If smoke tests fail we automatically roll the change back and open a ticket with the failing test outputs and sample rows for debugging.

Treat AI proposals as first-class outputs — reviewable, explainable and runnable in isolated sandboxes so engineers can reproduce and reason about fixes fast.

From our pilots

Self-healing flows reduced incident volume and **improved mean time to recovery (MTTR) by roughly 40%.** QA agents generate the smoke tests and example rows, and every remediation, test result and evidence bundle is logged in the metadata catalog so downstream owners can audit or revert changes.

Final Notes

We've been running AI multi-agent workflows in production for about two years. What we already know is that they're worth it: we stopped spending so many cycles on repetitive tasks and got those hours back for design and analytics work.

That said, AI isn't magic. Each stack needs its own knobs: what works for one environment won't drop unchanged into another. Expect to set confidence thresholds, wire in smoke tests and validation suites, log provenance, and keep data engineers in the loop for edge cases and policy decisions.

So, how to start a pilot?
Below are 4 steps that worked for us:

1. **Pick a domain**

2. **Set metrics and capture a baseline**

   (time-to-first-draft, mapping precision, auto-apply rate and MTTR)

3. **Run shadow/validation mode**

   (generate ERDs, mappings and transforms, compare against the baseline and domain review)

4. **Gate and scale**

   Use confidence thresholds, smoke tests and a rollback path; expand auto-apply only as metrics and reviews prove stable.

DEDICATTED

Need help with architecture, tooling, or just an experienced data engineer team to run alongside you?

Reach out to us at Dedicatted. We'll show you how we actually plug this into data-modeling processes and help stand up a focused, low-risk pilot.

Contact Us

contact@dedicatted.com

+1 437 427-1824

Canada

208-25 Telegram Mews, Toronto, Ontario, M5V 3Z1

Poland

Krakusa 11, Cracow,30-535

Ukraine

Heroiv UPA St, 73J Lviv, 79000