AI Schema Harmonizer

Why This Matters

Every organization that uses multiple SaaS tools faces the same problem: similar data, incompatible formats. Your CRM stores customers one way, your support tool another, your billing system a third. Your observability stack has metrics in Prometheus format, logs in JSON, traces in Jaeger format, and alerts from PagerDuty. Your analytics pipeline ingests events from Segment, Mixpanel, and custom instrumentation, each with different schemas.

The traditional solution is to write custom ETL pipelines for each source. This works but scales poorly. Every new tool requires new transformation code. Schema changes break pipelines silently. The mapping logic lives in scattered scripts that nobody fully understands.

What if an AI could analyze multiple data sources, identify semantically equivalent fields across different schemas, and propose a normalized target schema? Not just syntactic matching (field names that look similar) but semantic understanding: recognizing that Salesforce's "Account" is HubSpot's "Company" is Stripe's "Customer", even though the field structures differ completely.

The Product Vision

Users provide sample data from multiple sources, either as JSON/CSV files, API responses, or schema definitions. The AI analyzes the samples, identifies entities and relationships, maps semantically equivalent fields across sources, and proposes a unified schema.

The output is not just a schema but a transformation specification: for each source, here is how to map its fields to the normalized schema. Where mappings are ambiguous, the AI flags them for human review rather than guessing. Where data types differ (one source stores dates as ISO strings, another as Unix timestamps), the AI proposes conversion functions.

The system should handle:
- Direct mappings: `source.email` → `normalized.email_address`
- Renamed fields: `source.acct_name` → `normalized.account_name`
- Nested to flat: `source.address.city` → `normalized.city`
- Computed fields: `normalized.full_name` = `source.first_name` + " " + `source.last_name`
- Type conversions: Unix timestamp → ISO 8601
- Enum mappings: `source.status` ("A", "I", "P") → `normalized.status` ("active", "inactive", "pending")

Domain Focus: Observability Data

While the approach generalizes, a focused proof of concept should target a specific domain. Observability data is particularly interesting because:

Multiple competing formats exist. Prometheus metrics use a specific exposition format. CloudWatch has its own structure. Datadog, New Relic, and Grafana Cloud each have proprietary schemas. OpenTelemetry is emerging as a standard but adoption is incomplete.

Semantic equivalence is non-trivial. A "request latency" metric might be called `http_request_duration_seconds` in Prometheus, `HTTPRequestLatency` in CloudWatch, and `trace.duration` in Datadog. The units might differ (seconds vs milliseconds). The label/dimension names vary (`endpoint` vs `path` vs `route`).

Real business value exists. Organizations migrating between observability platforms, or running multi-cloud with different monitoring tools, desperately need this capability. The alternative is months of manual mapping work.

Open standards provide ground truth. OpenTelemetry defines semantic conventions for metrics, logs, and traces. The AI's proposed schema can be evaluated against these conventions, providing an objective measure of quality.

Agentic Architecture

Schema Discovery Agent ingests sample data and infers schemas. For JSON, it handles nested structures, arrays, and mixed types. For CSV, it infers column types and handles messy real-world data (inconsistent quoting, mixed delimiters). It produces a structural representation of each source's schema.

Entity Recognition Agent identifies the core entities in each schema. In observability data: metrics, logs, traces, spans, alerts. In CRM data: contacts, companies, deals, activities. It uses LLM reasoning to understand what each field represents semantically, not just syntactically.

Mapping Agent finds correspondences between entities and fields across sources. This is the core AI challenge. It must recognize that `prometheus.http_requests_total` and `cloudwatch.RequestCount` represent the same concept despite different names, structures, and semantics (counter vs gauge, rate vs absolute).

Schema Synthesis Agent proposes a unified target schema based on the identified mappings. It resolves conflicts (when sources disagree on types or granularity) by choosing the most expressive representation or flagging for human decision.

Transformation Generator produces executable transformation code for each source. This could be SQL, Python/Polars, dbt models, or a declarative transformation spec. The transformations should be human-readable and auditable.

Implementation Hints

Start with structural analysis before semantic. Parse the schemas, identify data types, detect nested structures, find primary keys and relationships. This structural understanding constrains the semantic matching problem.

Use embeddings for initial field matching. Embed field names and sample values, then find nearest neighbors across schemas. This catches obvious matches (email to email) quickly. Reserve LLM reasoning for ambiguous cases.

Build a domain-specific ontology. For observability, catalog common metric types (counters, gauges, histograms), standard dimensions (service, environment, region), and known naming conventions. This prior knowledge dramatically improves matching accuracy.

Handle the long tail of edge cases. Real schemas have weird fields that do not map cleanly. The system should explicitly mark "unmapped" fields rather than forcing bad mappings. Some fields legitimately have no equivalent in other sources.

Generate confidence scores. Not all mappings are equally certain. `email` → `email_address` is high confidence. `acct_mgr_id` → `account_manager_id` is medium. `x7_flag` → ??? is low. Surface these scores so humans know where to focus review.

Test with real messy data. Academic datasets are too clean. Get actual exports from Salesforce, HubSpot, and Stripe. Get real Prometheus and CloudWatch metrics. The edge cases in production data will break naive approaches.

Validation Approach

OpenTelemetry Semantic Conventions provide a benchmark. Take metrics/logs/traces from Prometheus, CloudWatch, and Datadog. Have the AI propose mappings to OpenTelemetry format. Compare against the official semantic conventions. Measure precision and recall of field mappings.

CRM Migration Datasets exist from vendors. HubSpot and Salesforce publish migration guides that implicitly define field mappings. These can serve as ground truth for evaluating CRM schema harmonization.

Synthetic Multi-Source Data lets you control the ground truth. Generate fake customer data, then transform it into Salesforce-like, HubSpot-like, and Stripe-like formats with known mappings. Test whether the AI recovers the original mappings.

Human Evaluation is ultimately necessary. Have data engineers review proposed mappings and rate them as correct, incorrect, or ambiguous. This catches semantic errors that automated metrics miss.

Datasets for Validation

Several established benchmarks exist for schema matching evaluation:

Valentine ([delftdata/valentine](https://github.com/delftdata/valentine)) is the most directly applicable benchmark. It provides an extensible experiment suite for schema matching methods with fabricated dataset pairs containing ground truth mappings. The [valentine-data-fabricator](https://github.com/delftdata/valentine-data-fabricator) generates dataset pairs classified into four categories: unionable, view-unionable, joinable, and semantically-joinable. Download the datasets-archive and use the ground truth to measure precision/recall of your AI's mappings.

OAEI (Ontology Alignment Evaluation Initiative) at oaei.ontologymatching.org has run for 20+ years evaluating ontology matching systems. Their datasets include the "anatomy" track (matching Adult Mouse Anatomy to NCI Thesaurus) and conference domain ontologies. While more academic than practical, OAEI provides rigorous benchmarks with established baselines.

OpenTelemetry Demo Dataset ([openobserve/opentelemetry-demo-dataset](https://github.com/openobserve/opentelemetry-demo-dataset)) generates production-like observability data from a 16-service e-commerce application. It produces logs from 20 sources, metrics (host/container/application/database), and traces with 7 distributed flows. Run with Docker Compose to generate OTLP-format telemetry for testing observability schema mapping.

Prometheus Benchmark Tools help generate metrics data:
- [VictoriaMetrics/prometheus-benchmark](https://github.com/VictoriaMetrics/prometheus-benchmark) generates production-like workload using real node_exporter metrics (~1230 unique metrics per target)
- [little-angry-clouds/prometheus-data-generator](https://github.com/little-angry-clouds/prometheus-data-generator) lets you define custom metric names, types, labels, and value sequences

OpenTelemetry Semantic Conventions at opentelemetry.io/docs/specs/semconv/ define the target schema for observability data. Map Prometheus/CloudWatch/Datadog formats to these conventions and measure how well your AI's mappings align with the official specifications.

Practical approach: Start with Valentine for general schema matching evaluation. Use the OpenTelemetry demo + Prometheus generators to create multi-format observability data with known semantic equivalences. Evaluate against OpenTelemetry semantic conventions as ground truth.

Interesting Questions

How do you handle schema evolution? Real schemas change over time. A mapping that was correct last month may be wrong now. Can the AI detect when schemas have drifted and flag mappings for re-review?

What is the right level of normalization? Sometimes denormalized schemas are more practical. Should the AI propose multiple target schemas at different normalization levels and let users choose?

How do you handle lossy mappings? Some sources have richer data than others. Mapping from a detailed source to a sparse source loses information. Should the AI warn about information loss, or propose auxiliary tables to preserve it?

Can the system learn from corrections? When a human corrects a mapping, can that feedback improve future suggestions? This is few-shot learning in a structured domain, an interesting ML problem.

How do you handle private/sensitive fields? PII appears differently across systems (SSN, national_id, tax_id). The AI needs to recognize these as sensitive and handle them appropriately, perhaps proposing consistent masking or encryption.

Broader Value

Data integration is one of the oldest problems in computing, and it remains unsolved. Every data warehouse project spends enormous effort on schema mapping. Every company acquisition triggers months of data integration work. Every SaaS migration requires painful field-by-field mapping.

If AI can meaningfully automate even part of this work, the productivity gains are substantial. The question is whether current LLMs have sufficient semantic understanding to handle the ambiguity inherent in real-world schemas. This project is a direct test of that capability.

The observability focus also connects to the broader trend of AI-assisted operations. If an AI can understand what metrics and logs mean well enough to normalize them, it is partway to understanding them well enough to diagnose issues. Schema harmonization is a stepping stone to more ambitious AIOps capabilities.