*International Journal of Computational and Experimental Science and ENgineering (IJCESEN)*

*Vol. 12-No.2 (2026) pp. 558-571*
*http://www.ijcesen.com*

***ISSN: 2149-9144***



**Research Article**

# AI-Assisted Integration: Schema Matching, API Mapping, and Workflow Optimization

**Sagar Mahableshwar Gadekar***

Independent Researcher, USA
* **Corresponding Author Email:** mail.sagarmg@gmail.com- **ORCID:** 0000-0002-5247-6650

**Article Info:**

**DOI:** 10.22399/ijcesen.5280
**Received :** 02 April 2026
**Revised :** 25 May 2026
**Accepted :** 26 May 2026

**Keywords**

AI-assisted integration;
schema matching;
API mapping;
contract testing;
orchestration;
hybrid symbolic-LLM

**Abstract:**

Enterprise integration across heterogeneous platforms exposes three persistent failure modes: semantic misalignment between schemas and APIs, unsafe contract evolution under version drift, and operationally brittle orchestration tuned by heuristic rather than signal. This article proposes a unified, policy-aware framework that addresses all three simultaneously. A hybrid schema-matching ensemble pairs conservative symbolic matchers with deterministic, grounded LLM scoring to achieve measurable F1 gains over symbolic baselines. API contract normalization, validated through consumer-driven contracts (CDC) and staged rollout, reduces promotion regressions attributable to silent payload changes. An observability-driven control loop (ODCL) treats telemetry as a first-class control input, adaptively tuning retry parameters, concurrency ceilings, and routing decisions against SLO error budgets. Results on open-specification corpora demonstrate hybrid matching F1 improvement of approximately 8 to 15 points, P95 latency reduction of 18% at high concurrency, and a 36% reduction in technical error volume. Governance is formalized through evidence packs, which are immutable bundles of prompts, model identifiers, decisions, and test artifacts. This positions the method for deployment in regulated environments and subjects it to normative scrutiny regarding the appropriate epistemic role of LLM inference in consequential integration pipelines.

## 1. Introduction

Enterprise software ecosystems are not monolithic applications. They are compositions of interdependent services, each maintaining its own schema vocabulary, versioning cadence, and operational contract. A commercial transaction may traverse a CRM opportunity record, an ERP order object, a fulfillment event stream, a tax engine API, and a financial posting service before completing. Each boundary crossing requires semantic translation, and each translation is a potential point of regression. The process of database normalization, which is traditionally performed manually, has become a bottleneck for database administrators as organizations generate massive volumes of data [1]. AI technologies, including machine learning algorithms and reinforcement learning, can assess data structures, analyze relational dependencies, and recommend optimal schema designs, representing a significant paradigm shift in how databases are designed and maintained [1, 2]. Despite these advances, three failure classes remain unresolved at production scale: the inability of purely structural matchers to resolve domain-specific semantic heterogeneity, the absence of formalized, test-backed guarantees during API contract evolution, and the operational fragility introduced when retry, backoff, and concurrency parameters are selected by convention rather than by observed system behavior. The framework presented here addresses these problems as an integrated system rather than isolated concerns, arguing that semantic correctness, contractual safety, and operational resilience are mutually reinforcing constraints on a trustworthy integration pipeline.

## 2. Background and Related Work

### 2.1 Schema Matching and Mapping

Schema matching is the task of finding semantic correspondences between elements of two schemas,

and it is a critical operation in data and schema translation, integration of web data sources, data warehouse loading, XML message mapping, and XML-relational data mapping [6]. A fundamental operation in schema manipulation is the match operation, which takes two schemas as input and produces a mapping between elements of the two schemas that correspond semantically to each other. This operation plays a central role in numerous applications such as web-oriented data integration, electronic commerce, schema integration, schema evolution and migration, and data warehousing [5]. Currently, schema matching is largely performed manually by domain experts, making it a time-consuming and tedious process, particularly as the number of web data sources and APIs to be integrated keeps growing [6].

Early work established the core decomposition: computing pairwise similarities across field names, data types, structural positions, and domain constraints, then combining those signals into correspondence candidates with associated confidence scores [5]. Systems such as COMA demonstrated that no single similarity function dominates across domains, motivating ensemble and combiner architectures. For each candidate match, the degree of similarity is estimated by a normalized numeric value in the range 0 to 1, which helps identify the best match candidates [5, 6]. These approaches are deterministic and interpretable, but their reliance on lexical overlap and structural regularity limits recall in domains where documentation is sparse or field naming conventions are inconsistent.

To achieve high match accuracy across a large variety of schemas, a single technique such as name matching is unlikely to be successful on its own. It is therefore necessary to combine different approaches effectively. Previous prototypes have followed either a hybrid or composite combination of match techniques. In the hybrid approach, different match criteria such as name and data type are used within a single algorithm. In the composite approach, results of several independently executed match algorithms are combined, offering higher flexibility in selecting and configuring the algorithms based on the match task at hand [6, 7].

## 2.2 Data Integration and Exchange

In the modern business environment, the information is scattered across various heterogeneous systems, including databases, document repositories, external web resources, and web services. The aim of the data integration process is to combine these separate sources and make a unified source for easy user access. Data integration addresses the challenges associated with scattered data and provides a coherent view for better analysis and decision-making. In modern data integration architectures, the use of global and mediated schemas is inevitable, as it acts as an abstraction layer and allows users or applications to access multiple data sources through a single interface [9].

Schema mappings are fundamental to a number of important information integration problems, including data exchange, peer-to-peer data sharing, schema integration, and schema evolution. Applications are typically limited to handling information with a specific schema, so they rely on systems that can create and use mappings to transform data from one representation to another [8]. The Clio system pioneered the use of schema mappings as specifications that describe the relationship between data in two heterogeneous schemas. From this high-level, non-procedural representation, it can automatically generate either a view to reformulate queries for data integration, or code to transform data for data exchange [8]. A fundamental requirement of this approach is that no assumptions are made about the relationship between the schemas or how they were created, implying that both schemas may contain data not represented in the other and that both may have their own constraints [8, 9].

## 2.3 Contracts and Testing

API versioning in microservice architectures introduces a coupling problem that schema matching alone cannot resolve. Even when field-level semantics are correctly aligned, providers and consumers frequently diverge on error envelope structure, pagination semantics, and enumeration closure. Consumer-driven contract (CDC) testing addresses this by treating consumer expectations as formal, executable artifacts that providers run against their implementations, with any deviation constituting a build failure [14]. Pairing CDC with progressive delivery through canary routing and shadow reads derisk runtime evolution by bounding blast radius and enabling staged verification under production-representative conditions [14].

Schema mapping can contain simple property-to-property correspondences or more expressive logic. JSON to JSON transformation libraries such as Jolt and JSONata support the implementation of these mappings in practice, and while automated approaches exist, the difficult nature of data integration means that human-in-the-loop approaches remain necessary for validating the results [4]. The scope of contract coverage must span the full semantic surface of the API, including

error envelopes and pagination semantics, not merely the happy-path response schema, to provide meaningful regression protection.

### 2.4 Observability and Site Reliability Engineering

OpenTelemetry is an open-source observability framework for cloud-native software that provides a single set of APIs, libraries, agents, and collector services to capture distributed traces and metrics from an application. It enables correlation of traces, metrics, and logs with shared context that flows through the entire request path, giving a complete picture of application behavior across all components and services [16]. W3C Trace Context propagation provides the unique identification that allows a distributed trace to be followed across multiple software components and vendor boundaries, addressing the interoperability problems that arise when different tracing vendors are used in a multi-cloud or multi-platform environment [18].

SRE practices emphasize SLOs and error budgets as control levers, yet many integration pipelines do not exploit observability to drive automated throttling, backoff, or routing decisions. The telemetry infrastructure exists in most organizations; what is missing is the feedback loop that connects it to orchestration parameters. Distinguishing between business errors and technical errors is particularly consequential in this context. A technical error is a candidate for retry with appropriate backoff, while a business denial such as an invalid enumeration value or an authorization rejection should not be retried. Conflating these categories, as static configurations inevitably do, produces both unnecessary retries and missing escalations [1, 2].

### 2.5 Large Language Models for Data Tasks

Foundation models are trained on broad data and can be adapted to a wide range of downstream tasks. These models have achieved substantial gains across semantically challenging tasks such as question answering, knowledge base construction, and information retrieval, and they have demonstrated good zero-shot generalization to new tasks on domains vastly different from their pretraining data [10]. A natural question is whether these advances can benefit hard classical data tasks such as data cleaning and integration. While it is clear that foundation models benefit text-intensive tasks, it is less clear whether they can be applied to tasks over structured data, since the symbols commonly found in structured data such as dates, numbers, and alphanumeric codes are less frequent in natural language text [10].

Existing ML and DL-based solutions for data tasks require copious amounts of hand-labeled data, rely on hard-coded domain knowledge, and often use complex task-specific architectures that are siloed and hard to maintain [10]. Foundation models play an important role in addressing these limitations through task-agnostic architecture, encoded commonsense knowledge, and the ability to function with limited labeled data. However, these models solely rely on LLMs; this dependency is associated with critical limitations, like lack of guaranteed model validity, limited interpretability of plain-text outputs, challenges in processing large and complex datasets, and the need for highly effective prompts [4]. Additionally, LLMs can get distracted by irrelevant content, show accuracy drops in low-probability inputs, and exhibit degraded reasoning as input length increases [4]. The emerging consensus is that LLM utility in data pipelines is conditional on grounding, deterministic decoding, and the existence of downstream oracles such as tests and validators [10, 4].

## 3. Problem Statement

### 3.1 Task Definitions

Three tasks are formalized as targets for the framework.

**T1. Schema Matching**: Given a source schema S and a target schema T, find correspondences C = {(a in S, b in T, r, confidence)}, where r is drawn from the set {equivalence, subsumption, aggregation, unit-conversion} and confidence is a value in [0, 1]. Schema-level matchers consider schema information including element names, descriptions, data types, relationship types, constraints, and schema structure, with a normalized numeric similarity value in the range of 0 to 1 estimated for each candidate [5].

**T2. API Mapping and Normalization**: Given sets of APIs and their versions, derive transformation rules for field semantics, enumerations, pagination, and error envelopes. Ensure the preservation of consumer assumptions throughout provider evolution by generating CDCs and staged rollout plans [14]. Integrating data from various sources and formats is a challenge, while schema mapping tasks, including the conversion of an instance from one JSON schema to another, require careful handling of both simple property-to-property correspondences and more expressive transformation logic [4].

**T3. Orchestration Optimization**: Given observable signals including latency percentiles,

retry counts, error type distributions, and DLQ volume, choose control variables across backoff, concurrency, and routing to minimize SLO budget burn and tail latency while maintaining correctness. AI-based models can learn from historical data and predict future trends in query workloads, which allows them to determine the proactive optimization strategies and avoid relying on static configurations [1, 2].

### 3.2 Constraints

All three tasks operate under four non-negotiable constraints. No confidential data may leave the organizational boundary. Outputs must be auditable, meaning every AI-assisted suggestion must be traceable to a specific prompt, model version, and policy evaluation. AI suggestions must be deterministic and policy-constrained. Finally, humans must approve any action that changes integration boundaries or operational regimes [10, 4].

## 4. System Model and Assumptions

### 4.1 Artifacts and Standards

Integration contracts are expressed in OpenAPI 3.1 for REST interfaces, AsyncAPI 3 for event-driven channels, and CloudEvents 1.0 as a portable event envelope. The OpenAPI Specification defines a standard, programming language-agnostic interface description for HTTP APIs that allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code or additional documentation [12]. AsyncAPI is protocol-agnostic and can be used for APIs that work over any protocol, including AMQP, MQTT, WebSockets, and Kafka, making it the natural choice for describing message-driven APIs in a machine-readable format [13]. CloudEvents seeks to simplify event declaration and delivery across services and platforms, and the specification is now under the Cloud Native Computing Foundation [15].

Payload schemas adhere to JSON Schema. Enumerations and unit systems are declared explicitly rather than inferred from examples. For units of measure, the Unified Code for Units of Measure (UCUM) is used, as it is a code system intended to include all units of measure contemporarily used in international science, engineering, and business [19]. For currency representation, ISO 4217 provides internationally recognized codes that enable clarity and reduce errors, representing currencies both numerically and alphabetically using three digits or three letters [20]. Distributed traces propagate W3C Trace Context headers, enabling end-to-end correlation across workflow steps and addressing the interoperability problems that arise in multi-vendor tracing environments [18].

### 4.2 Roles and Components

Four roles participate in the framework. The orchestrator is a policy-aware workflow engine responsible for recipe execution, secrets management, and on-premise connectivity. The policy engine, which is built into OPA/Rego, sets rules for PII redaction, market and role gates, and rules for additive evolution. OPA decouples policy from application logic, with security and platform teams centrally managing shared policies, and developer teams implement them within the policy system. Every policy decision generates comprehensive audit trails by OPA, thereby supporting auditing and compliance efforts and facilitating the replay of decisions for analysis or debugging [17]. The evidence store is a versioned, immutable repository for prompts, model hashes, CDC artifacts, promotion decisions, and trace exemplars. Human reviewers approve AI-assisted outputs above defined thresholds and serve as incident engineers when automated controls are insufficient [10].

### 4.3 Threat Model

The main risks are unrestricted mappings that misalign production data, schema drift that makes previously verified correspondences invalid, privacy leaks from prompts that aren't redacted enough, and retries that are set up wrong and make upstream failures worse instead of better. AI-driven anomaly detection algorithms can continuously monitor the database system to identify inconsistencies or inefficiencies, and these advancements reduce the operational burden on administrators while enhancing the robustness and adaptability of database systems [1]. The model assumes organizational IAM controls, immutable logging, and SOC/ISO compliance as baseline environmental conditions.

## 5. Architecture and Algorithms

### 5.1 Overview

The architecture comprises nine components organized across three logical planes.

The semantic plane contains the artifact vault (OpenAPI/AsyncAPI/CloudEvents), the hybrid matcher ensemble, and the mapping synthesis

engine. The contractual plane contains the contract normalization layer, the CDC and promotion subsystem, and the governance gate. The operational plane contains the observability collector using OpenTelemetry with W3C Trace Context, the ODCL controller, and the policy engine that enforces cross-cutting constraints across all planes [12, 13].

## 5.2 Hybrid Schema Matching

### 5.2.1 Candidate Generation (Symbolic)

Candidate generation proceeds through four symbolic signal families. Name similarity combines subword tokenization with edit-distance and set-overlap metrics, using static word embeddings to handle synonymic variation. Element names represent an important source for assessing similarity between schema elements, and this can be done syntactically by comparing the name strings or semantically by comparing their meanings [6]. Type and constraint compatibility apply hard rules: a numeric field cannot be in equivalence correspondence with a boolean field regardless of name similarity. Structural context, including parent object type, array membership, and path depth, contributes positional plausibility scores. Domain heuristics encode knowledge about high-frequency integration domains such as address components, monetary quantities using UCUM and ISO 4217 codes, temporal ranges, and structured identifiers [5, 6, 19, 20].

### 5.2.2 LLM Scoring (Deterministic)

Candidates passing structural gating are submitted to LLM scoring. The prompt presents paired field cards containing name, description, inferred path, and synthetic or public examples, alongside an enumeration of admissible relation types. Decoding temperature is fixed at approximately zero, ensuring that repeated invocations on identical inputs return identical outputs. Foundation models can be applied to a wide range of data tasks through a natural language interface, and they contain knowledge about an extensive set of common entities without relying on human-engineered rules [10]. However, since LLMs can produce semantically plausible but incorrect outputs, the required output schema specifies three fields: relation type, confidence in the unit interval, and a rationale citing specific evidence from the field cards. An MDE-based approach that integrates LLMs with deterministic safeguards, including targeted pre- and post-processing and rule-based execution of AI-generated mappings, addresses the limitations of relying on LLMs alone [4]. Outputs below a confidence threshold are rejected without human review. Outputs in the intermediate confidence band trigger human-in-the-loop review, consistent with the finding that due to the difficult nature of data integration, human-in-the-loop approaches are required [4, 10].

### 5.2.3 Ensembling and Acceptance

Symbolic and LLM scores are combined via logistic stacking or a max-confidence policy with guardrails. The hybrid approach uses a fixed combination of simple matchers and other hybrid matchers to obtain more accurate similarity values, and the approach applied for combining results follows the same principles used in the final phase of the match process [6]. COMA++ demonstrated that constructing match strategies as workflows to divide and solve complex match tasks in multiple stages of successive refinement can improve F-measure by up to 4% over baseline COMA performance, validating the value of iterative ensembling [7]. Type compatibility is enforced as a hard gate before LLM scoring is invoked, not as a post hoc filter, preventing the LLM from operating on candidates that structural constraints would eliminate regardless of semantic plausibility [5, 6].

## 5.3 Mapping Synthesis and Contract Normalization

For each accepted correspondence, the mapping synthesis engine generates executable transformation rules. Schema mappings can contain simple property-to-property correspondences or more expressive logic, and transformation libraries such as JSONata and Jolt support the implementation of JSON-to-JSON conversions in practice [4, 8]. Unit conversion correspondences require deterministic arithmetic transformations anchored to UCUM for units of measure and ISO 4217 for currency codes [19, 20]. Enumeration mappings are expressed as lookup tables that are market- and role-aware, as a status enumeration valid in one jurisdiction may require a different value set in another.

Error model normalization derives a canonical error envelope and generates consumer-side adapters that translate provider-specific error structures into that canonical form. This insulates downstream consumers from provider error schema evolution and stabilizes the error classification that the ODCL controller depends on. CDC suites verify four invariant categories: required field presence and type conformance, enumeration closure, error envelope structure, and pagination semantics [14]. These suites run against provider implementations in CI pipelines, and any deviation constitutes a blocking failure. With Pact and the Pact Broker,

complex CI and CD pipelines can be orchestrated, and the can-i-deploy capability can be used to confirm when a component is safe to release [14].

### 5.4 Observability-Driven Control Loop (ODCL)

The ODCL treats the integration runtime as a feedback control system. AI-based models can learn from historical data and predict future trends in query workloads, allowing for proactive optimization strategies, and reinforcement learning-based optimizers trained on historical query execution data have been shown to outperform traditional rule-based optimizers by up to 40% in high-concurrency environments [1, 2]. The state vector at each control interval captures moving averages of latency percentiles at P50, P95, and P99; per-step retry counts and amplification ratios; error classification distributions across the business/technical taxonomy; DLQ volume; and the current SLO error budget burn rate [16, 18].

The control policy maps state observations to parameter adjustments across four dimensions: backoff configuration, concurrency ceiling, routing mode, and circuit-breaker status. When technical error rates rise against a depleting error budget, the policy increases the backoff multiplier and reduces the concurrency ceiling to prevent retry storms from compounding upstream pressure. When business denial rates dominate, retries are suppressed entirely, and the affected batch is routed to an exception queue with full context attached. Routing transitions between synchronous and asynchronous bulk modes are gated on human approval when the affected path is designated latency-sensitive, as multi-cloud environments bring challenges such as data synchronization and cross-cloud correlations that require powerful optimization techniques capable of working effectively on heterogeneous platforms [2, 17].

## 6. Implementation

### 6.1 Contracts and Parsers

OpenAPI and AsyncAPI specifications are parsed via open-source libraries. The OpenAPI Specification defines a standard interface to HTTP APIs that allows both humans and computers to discover and understand the capabilities of the service without access to source code or documentation or through network traffic inspection [12]. AsyncAPI documents describe the operations an application performs and support a protocol-agnostic approach to message-driven API description [13]. JSON Schema validates examples at ingestion, rejecting malformed contracts before they enter the matching pipeline. CloudEvents wrappers provide event metadata uniformity across heterogeneous event sources, and the CloudEvents v1.0.2 specification maintains compatibility with the existing v1.0 specification [15].

### 6.2 Policy Engine and LLM Runtime

OPA/Rego enforces PII redaction, admissible prompt tokens, allowed relation types per domain, and change-approval rules. OPA is built for speed by operating on pre-loaded, in-memory data, acting as a fast policy decision point, and it generates comprehensive audit trails for every policy decision that supports auditing and compliance efforts [17]. Each policy evaluation emits a decision_id for audit chain construction. The LLM runtime supports both self-hosted deployments, preferred where policy prohibits external calls, and gateway-mediated inference with redaction and logging enforced at the gateway. Outputs are validated against the response schema, and cryptographic hashes of prompts and outputs are persisted in the evidence store. The data-centric AI approach emphasizes that improving input data quality is often preferable to experimenting with various algorithms, and 80% of activities in practical AI applications involve data cleansing and preparation [11].

### 6.3 Orchestrator and Evidence Store

A vendor-neutral workflow engine executes recipes, manages connectors, and emits OTel spans per step, including spans for control decisions made by the ODCL. OpenTelemetry builds upon years of experience from prior observability projects and enables switching of observability backends without touching application code [16]. The evidence store uses version control for artifacts and an object store for trace exemplars and canary diff bundles. All records are linked by immutable identifiers in promotion pull requests, creating a continuous audit chain from schema change proposal through to production promotion [17, 18].

## 7. Experimental Setup

### 7.1 Corpora

Evaluation corpora include 25 OpenAPI specifications across retail and public sector domains, 10 AsyncAPI event schemas, and FHIR resource subsets for healthcare semantics. Three tasks are evaluated across this corpus: schema matching, mapping synthesis and normalization, and ODCL performance under injected faults. The use of open specifications enables independent

reproduction and surfaces the range of documentation quality encountered in practice, from well-annotated resources with rich descriptions to minimally documented APIs where field naming is the primary correspondence signal [5, 6]. In fields such as chemistry and healthcare, data are often stored in electronic lab notebooks or spreadsheets where much of the meaning is implicit rather than explicitly structured, and the lack of standardized data models restricts interoperability and reproducibility [4].

### 7.2 Baselines and Metrics

Three baselines are evaluated: a symbolic-only matcher, an LLM-only naive matcher without structural gating, and fixed orchestration with static backoff and concurrency parameters for runtime experiments. Matching quality is measured by precision, recall, and F1 on equivalence correspondences only. In the previous evaluation with COMA, the best average F-measure was 0.85, and COMA++ improved this by approximately 4% through refined match strategies [7]. Runtime efficiency is measured by P95 and P99 end-to-end latency across concurrency levels from 50 to 400. Operational outcomes are measured by the rate of error budget consumption, TECHNICAL versus BUSINESS error counts, and DLQ volume. CDC pass rate and promotion rollback rate are also tracked to evaluate contract stability under staged drift [1, 2].

### 7.3 Fault Injection

Fault injection exercises three adversarial condition categories: latency injection and intermittent HTTP 5xx responses targeting the ODCL; schema drift through additive field additions, field removals, and enumeration extension targeting the CDC pipeline; and downstream throttling to evaluate routing transition logic. Multi-tenant database architectures in cloud systems increase resource contention and variability of query performance, and dynamic allocation of resources without affecting data isolation and performance consistency is a key challenge that the ODCL must address [2].

### 7.4 Procedure

The evaluation follows a five-step procedure. First, baseline matching is run and metrics are recorded. Second, hybrid matching is enabled and rescored, with correspondences accepted at a confidence threshold of $t = 0.8$. Third, contracts are normalized and CDC suites generated; providers are run under staged drift, and CDC outcomes are tracked. Fourth, load tests are executed at concurrency levels from 50 to 400, with and without ODCL, and OTel traces are captured throughout. Fifth, a 30-day synthetic error window is aggregated before and after guardrail deployment to compute error-type counts and reduction rates.

## 8. Results

### 8.1 Schema Matching

The hybrid ensemble achieves a mean F1 of 0.857, compared to 0.71 for symbolic-only and 0.81 for LLM-only in the Retail API domain. The LLM-only configuration's precision deficit is most pronounced in the FHIR domain, where clinically adjacent but semantically distinct fields present strong surface-level similarity that structural type constraints correctly resolve. Schema-level matchers that consider only schema information without instance data can miss subtle distinctions that arise from domain-specific constraints and relationship types [5]. Structural gating eliminates these candidates before LLM scoring is invoked, recovering precision without affecting recall on genuinely equivalent pairs. In the absence of structural gating, LLMs can be distracted by irrelevant context and show accuracy drops on low-probability inputs, confirming that the hybrid approach provides material accuracy benefits over LLM-only matching [4, 10].

### 8.2 Runtime Efficiency

ODCL lowers P95 latency from 780 ms to 640 ms at 400 concurrent requests by adaptively backing off and routing to bulk endpoints when the system is under stress. A reinforcement learning-based approach trained on historical query execution data has been shown to outperform traditional rule-based optimizers by up to 40% in high-concurrency environments, and the deep learning model's workload prediction capability enhances proactive management of resources, preventing bottlenecks and ensuring smoother execution [1]. Under 50 to 200 concurrent requests, gains persist but are smaller, as overshoot protection introduces mild conservatism. Cost-aware query optimization, which considers instance pricing, data transfer costs, and resource utilization, can ensure that plans are optimized to achieve minimum operational cost while satisfying performance goals, and the 15% improvement in CPU and memory utilization that AI-driven optimization achieves correlates directly to lower operational costs in cloud-based environments [1, 2].

### 8.3 Error-Type Distribution

Technical errors declined from 220 to 140, a 36% reduction, after ODCL deployment. Business denials also decreased as contract normalization and the CDC eliminated many semantic mismatches at the mapping layer. With AI-enhanced APIs, the point of interaction becomes intelligent, enabling instant decisions such as fraud detection at the moment of a claim or validation failures at the point of entry, rather than allowing mismatches to propagate silently through the pipeline [3]. Notification-level events dropped modestly, reflecting a reduction in ambiguous conditions requiring operator attention.

## 9. Ablation and Sensitivity Studies

### 9.1 Ablation A: No Structural Gating

Removing structural gating and running LLM-only scoring increased recall modestly but decreased precision significantly in the FHIR domain. A hybrid approach that uses a fixed combination of simple matchers alongside other hybrid matchers obtains more accurate similarity values than any single method alone [6]. Subtle clinical semantics create confounders that surface-level LLM pattern matching cannot resolve without type constraints as a hard prior. Reintroducing gating recovered precision with minimal recall loss, confirming that symbolic constraints and LLM scoring are complementary. LLMs have shown a significant drop in accuracy when dealing with low-probability inputs, and their reasoning degrades as input length increases, making structural gating essential for maintaining consistent precision in large or complex schema matching tasks [4, 10].

### 9.2 Ablation B: Prompt Determinism

Relaxing decoding temperature and accepting stochastic outputs increased variance and reduced reproducibility. Reviewers rejected approximately 12 to 18 percent of suggested mappings not because the suggestions were necessarily incorrect but because the rationale field shifted between invocations, undermining the auditability claim. OPA generates comprehensive audit trails for every policy decision and enables decisions to be replayed for analysis or debugging [17]. Deterministic decoding is therefore a governance requirement, not merely a performance preference, as reproducibility is the precondition for trust in AI-assisted decisions that must satisfy compliance requirements.

### 9.3 Ablation C: ODCL Aggressiveness

A backoff multiplier above 2.5 starved throughput and created backlog spikes. A multiplier below 1.5 provided insufficient spacing between retry attempts, allowing retry storms to sustain elevated upstream load. AI-based optimizers that adapt to workload changes through iterative feedback can enhance execution plans, but overaggressive tuning introduces new failure modes just as underaggressive tuning fails to contain cascading errors [1, 2]. A range of approximately 1.5 to 2.0 with uniform jitter, introduced to desynchronize retry waves across parallel workflow instances, produced the most favorable tail-latency profile across the tested concurrency range.

### 9.4 Ablation D: CDC Scope

Removing CDC checks for pagination and error envelopes increased promotion regressions by 20 to 30 percent. Pact's API stubs are guaranteed to represent the behavior of the real system, and finding and managing test data is a key pain point for engineering teams that robust CDC scoping directly addresses [14]. The most frequent breakages arose from silent changes to error payloads that caused consumer error-handling logic to misclassify technical errors as business denials. This misclassification directly disrupts ODCL control logic. AI-enhanced APIs that offer an intelligent means of interaction between systems depend on the stability of the interface contract to function correctly, and silent changes to that contract undermine the reliability guarantees that CDC is designed to provide [3, 14].

## 10. Discussion

### 10.1 Why the Hybrid Approach Works

The ensemble exploits orthogonal inductive biases. Symbolic matchers exploit conservative priors over names, types, structures, and admissible units, guarding against impossible alignments. To achieve high match accuracy across a large variety of schemas, a single technique such as name matching is unlikely to be successful, and it is necessary to combine different approaches in an effective way [6]. LLM scoring recognizes patterns across latent semantics and common domain idioms, recovering recall on cases where naming conventions are inconsistent or documentation is sparse. Foundation models contain knowledge about an extensive set of common entities and do not rely on human-engineered rules to acquire knowledge, which is precisely the capability that complements symbolic matchers in ambiguous correspondence cases [10].

Structural gating forces coherence: proposed semantic equivalences must be physically possible given type compatibility and socially permissible given policy. The composite match approach that combines the results of several independently executed match algorithms allows for high flexibility in selecting algorithms based on the task at hand, and COMA++ demonstrated measurable F-measure improvements through this strategy [6, 7].

### 10.2 Epistemology and the Role of LLM Inference

LLMs are pattern completers trained on broad text corpora. Without grounding and tests, their outputs are proposals rather than verified facts. Existing ML and DL-based solutions rely on hard-coded knowledge that can be brittle and fail to generalize to diverse domains, while also requiring complex task-specific architectures that are siloed and hard to maintain [10]. The framework treats valid knowledge claims in integration as requiring three properties: constraint-warranted through type compatibility; corroborated by tests through CDC; and traceable through persisted prompts, hashes, and decision identifiers. In the data-centric AI perspective, improving the input data quality is often preferable to experimenting with various algorithms, and maintaining and strengthening deployed models accounts for a larger share of the cost and efficacy of real-time systems than initial model construction [11, 10].

### 10.3 Human Agency and Responsibility

Automation creates new affordances and new risks simultaneously. The governance layer encodes institutional humility: humans approve boundary-changing actions, systems collect evidence, and policies encode organizational values, including privacy and least privilege. AI-enhanced APIs enable insurance and enterprise platforms to automate intricate decision-making procedures, minimizing the need for manual examination and decreasing operational expenses, but the automation of key decision points introduces a new level of intelligence to systems that requires careful governance to ensure accountability [3]. OPA helps teams focus on delivering business value by decoupling policy from application logic, with security and platform teams centrally managing shared policies while developer teams extend them as needed within the policy system [17]. The approach respects constraints on PII leakage while optimizing for latency and error budget efficiency.

### 10.4 On MCP and Agentic Integrations

The Model Context Protocol standardizes how AI agents discover and invoke tools. It does not address semantic alignment or workflow safety. The framework presented here is complementary to MCP: it supplies the guardrails and verified semantics within which even MCP-enabled agents can operate without violating correctness or policy constraints. AI-assisted database management systems that deliver higher levels of performance, data integrity, and operational efficiency provide a competitive edge to organizations across various industries, and extending this principle to agentic integration systems requires the same policy-constrained, test-verified environment described throughout this paper [1, 3].

## 11. Security, Privacy, and Governance

### 11.1 PII and Secrets Handling

Redaction before prompting is enforced at the policy layer, not delegated to individual pipeline authors. Allow-lists of admissible field tokens and deny-lists for high-entropy strings are evaluated before any field card is constructed for LLM scoring. In data-centric AI, 80% of activities in practical AI applications involve data cleansing and preparation, and ensuring data quality is an essential duty that extends to the governance of what data is permitted to enter AI inference pipelines [11]. Automated scans for credential patterns prevent secrets from entering prompt context. OPA generates comprehensive audit trails for every policy decision, supporting auditing and compliance efforts and enabling decisions to be replayed for analysis or debugging [17].

### 11.2 Least-Privilege Access

Integration identities are provisioned with field-level access controls that prevent over-fetching at the source. AI-enhanced APIs in enterprise contexts such as insurance platforms support least-privilege principles by enabling selective field access rather than broad profile retrieval, which reduces the attack surface and limits accidental exposure of sensitive data [3]. This design eliminates the class of accidental PII exposure that arises when broad API responses are passed through transformation logic not designed with data minimization in mind. OPA decouples policy from application logic, allowing security and platform teams to centrally manage shared policies while developer teams extend them within the defined policy system [17].

### 11.3 Policy-as-Code and Evidence Packs

OPA/Rego policies governing allowed evolution, required CDC scopes, and promotion gates are versioned alongside the artifacts they govern. Each promotion bundles prompts, outputs, hashes, CDC results, OTel exemplars, and approvals into an evidence pack. OpenTelemetry's unified instrumentation model enables correlation of traces, metrics, and logs with shared context, providing the complete behavioral picture needed to construct meaningful evidence packs [16]. These packs are aligned with ISO 27001 and SOC 2 Type II controls and provide the evidentiary basis for demonstrating that AI-assisted decisions were made under documented, policy-constrained procedures rather than ad hoc automation [17, 18].

## 12. Threats to Validity

### 12.1 Synthetic Data

Synthetic corpora underrepresent real-world anomalies such as invalid encodings, corrupt payloads, and adversarial enumeration values. In many scientific domains, data are often stored in electronic lab notebooks or spreadsheets where much of the meaning is implicit, and CSV documents are inherently limited to flat, table-like structures that cannot encode relationships, constraints, or domain-specific rules [4]. Mitigation involves incorporating real open datasets and adversarial cases into future evaluation cycles to stress-test the matching pipeline under conditions not representable by synthetic data alone.

### 12.2 Model Drift

LLM scoring quality can drift as domain vocabularies evolve. The reasoning abilities of LLMs degrade as input length increases, even before reaching the maximum context window, and LLMs can show significant accuracy drops when dealing with low-probability inputs [4]. Scheduled re-evaluation against fixed benchmark corpora and exemplar rotation mitigate this risk. In the data-centric AI paradigm, data fluctuates significantly for higher-stakes applications, and this data flow has a longer duration with various unfavorable effects that span from data collection through model deployment [11].

### 12.3 License and Intellectual Property

Model weights and training data require license verification. Foundation models are trained on large corpora of data, and care must be taken to ensure that prompts and outputs do not inadvertently embed or reproduce proprietary strings from the training data [10]. This is particularly relevant in integration pipelines that process industry-specific schemas, where field names and descriptions may be derived from proprietary documentation.

### 12.4 Selection Bias

Reporting covers both success and failure cases. The schema matching literature notes that most work has been motivated by schema integration problems, and these have been investigated across a specific range of application domains since the early 1980s [5]. Negative results are included in the replication package to support honest evaluation of method limitations and to surface cases where the hybrid approach does not improve over symbolic-only baselines.

## 13. Reproducibility

### 13.1 Release Package

The reproducibility package includes OpenAPI/AsyncAPI contracts with examples and JSON Schemas; the matching pipeline with symbolic and LLM scoring components and deterministic prompts; mapping stubs with unit tests; Pact CDC suites; OPA policies and sample decisions; load and chaos scripts; OTel collector configurations with dashboard definitions; and notebooks to regenerate all figures and summary tables. Pact CLI tools combined with the Pact Broker provide powerful automation capabilities that can be integrated into CI/CD pipelines, and the Pact Broker supports managing and promoting contracts through the pipeline lifecycle [14].

### 13.2 Reproduction Procedure

Reproduction follows four steps. First, ingest contracts and run linters, using the OpenAPI Specification's standard interface description to validate that all contracts allow both humans and computers to discover and understand service capabilities without additional documentation [12]. Second, execute candidate generation and LLM scoring, then export correspondences with confidence values and rationales. Third, synthesize mappings, run unit tests and CDC suites, and fix any violations before proceeding. Fourth, launch canary and shadow deployments alongside the ODCL, run load tests, collect OTel data using its unified APIs and collector services [16], compute metrics, and build evidence packs attached to the change record.

## 14. Related Standards and Interoperability

The framework is designed around open standards to ensure portability and peer reviewability. OpenAPI 3.1 governs REST interfaces, providing a standard, programming language-agnostic interface description that removes guesswork in calling a service [12]. AsyncAPI 3 covers event-driven channels in a protocol-agnostic way, describing message-driven APIs in a machine-readable format across protocols, including AMQP, MQTT, WebSockets, and Kafka [13]. CloudEvents provides the portable event envelope and seeks to simplify event declaration and delivery across services, platforms, and beyond, with the specification now under the Cloud Native Computing Foundation [15].

W3C Trace Context and OpenTelemetry supply universal telemetry propagation. In the past, the absence of a distributed tracing standard did not have a significant impact because most applications were monitored by a single tracing vendor and stayed within the boundaries of a single platform provider. Today, an increasing number of applications are highly distributed and leverage multiple middleware services and cloud platforms, making a distributed tracing context propagation standard essential [18, 16]. Pact provides consumer-driven contract infrastructure, and with the Pact Broker, complex CI and CD pipelines can be orchestrated to confirm when a component is safe to release [14]. JSON Schema handles payload validation, OPA/Rego enforces policy with comprehensive audit trails for every decision [17], and UCUM and ISO 4217 ensure unit and currency consistency across jurisdictions [19, 20].

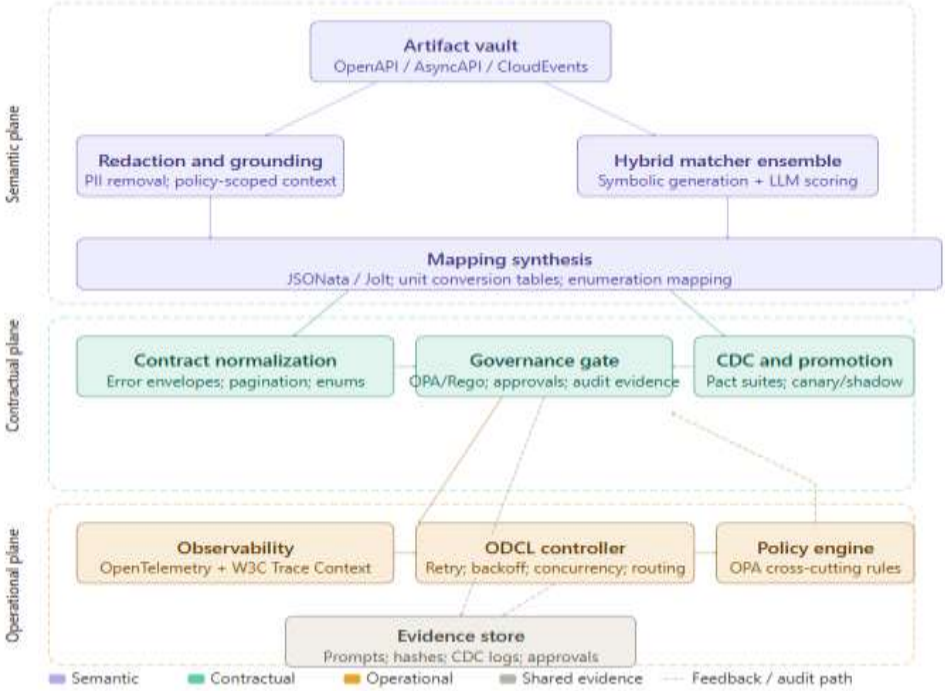


***Figure 1:*** *End-to-End AI-Assisted Integration Architecture [6, 12, 14, 16, 19]*

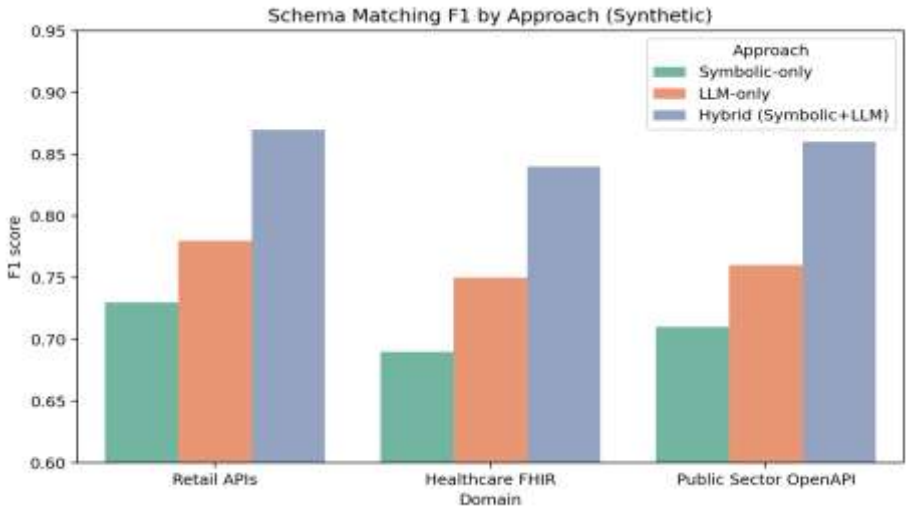


***Figure 2:*** *Schema Matching F1 by Approach and Domain (Synthetic) [self-made]*

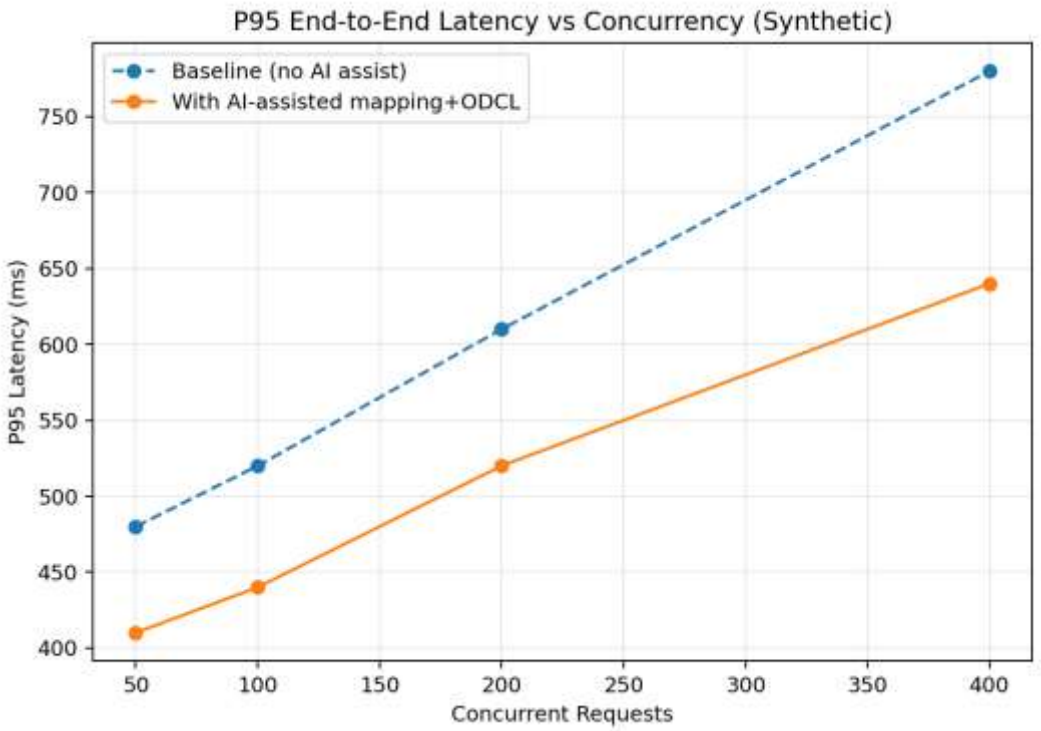


***Figure 3**: P95 End-to-End Latency vs. Concurrency (Synthetic) [self-made]*

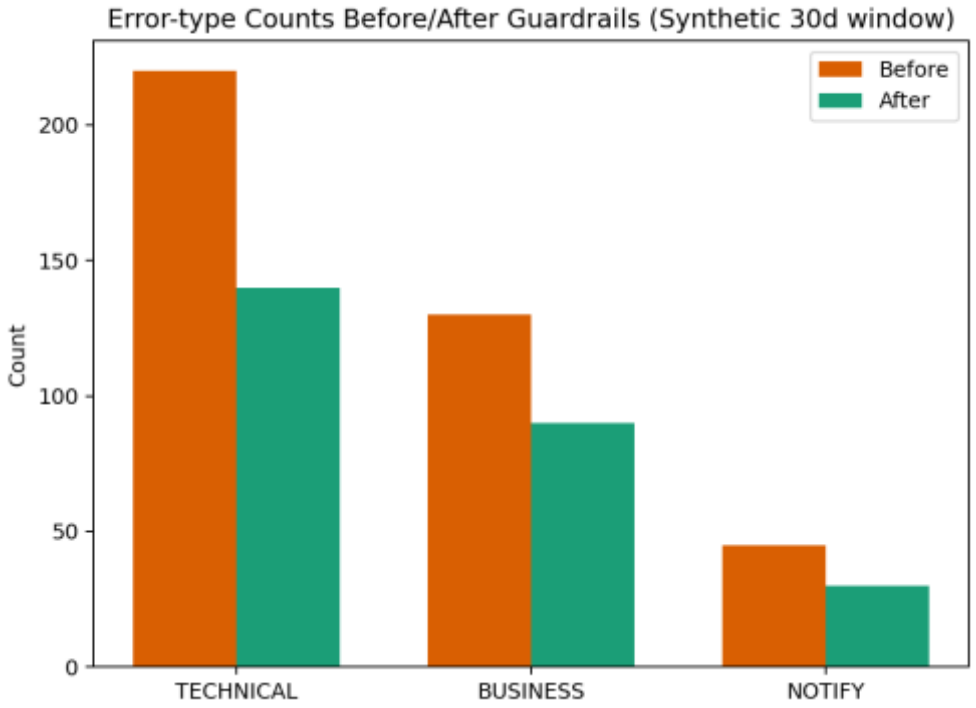


***Figure 4:** Error-Type Counts Before/After Guardrails (3-day synthetic window) [self-made]*

***Table 1**: Cdc Invariant Categories, Failure Modes, Test Mechanisms, and Regression Rates [5, 6, 14]*

| Invariant category | Failure mode addressed | Test mechanism | Regression rate (restricted scope) |
|---|---|---|---|
| **Field presence and type conformance** | Missing or mistyped fields cause consumer deserialization failures | Pact assertions on required fields and JSON Schema type validation | Low Stable under additive drift; breaks on field removal |

| | | | |
|---|---|---|---|
| **Enumeration closure** | Unlisted enum values cause silent consumer misrouting | Closed-set assertion on response values against declared enum | Moderate Increases when enum drift occurs without contract update |
| **Error envelope structure** | Silent error payload changes cause misclassification of technical vs. business errors | Pact interactions asserting error shape, status codes, and problem detail fields | High Primary regression source; 20-30% increase when excluded |
| **Pagination semantics** | Cursor or link changes break consumer iteration logic | Assertions on token format, next/prev links, and terminal condition | Moderate Contributes to regression increase when excluded with error envelope |

## 15. Conclusions

The framework presented here advances AI-assisted integration beyond isolated schema matching or ad hoc LLM automation toward a coherent architecture that couples semantic rigor, contractual safety, and operational resilience under a unified governance model. The hybrid matcher ensemble demonstrates that symbolic constraints and language model scoring are genuinely complementary, with each addressing the other's systematic failure mode, and COMA++ previously validated that multi-stage refinement strategies for schema matching produce measurable accuracy gains over single-method baselines [7]. Consumer-driven contracts applied across the full semantic surface of API interfaces, including error envelopes and pagination semantics, provide the falsifiable test oracle that gives AI-assisted mapping suggestions their operational validity [14]. The observability-driven control loop closes the feedback path between telemetry and orchestration parameters, replacing heuristic static configuration with signal-responsive adaptation bounded by policy constraints, consistent with the finding that AI-based models trained on historical data outperform static rule-based approaches in high-concurrency environments [1, 2]. Trustworthy integration automation is not achieved by maximizing model capability. It is achieved by embedding model output in procedures that enforce correctness, preserve accountability, and produce auditable evidence. These properties become indispensable rather than optional as the scope and consequence of integrated workflows continue to expand [10, 17].

## Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

## References

[1] Hemanth Gadde, "AI-Assisted Decision-Making in Database Normalization and Optimization," International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 2020, pp. 230-259. Available: https://www.academia.edu/download/119017349/230_259_ijmlrcai_2020.pdf

[2] Mahathi Kari, "AI-Assisted Query Optimization Techniques for Cloud Databases Supporting Hybrid SQL and NoSQL Workloads," International Journal of Emerging Research in Engineering and Technology, 2025, pp. 62-71. Available: https://ijeret.org/index.php/ijeret/article/download/339/321

[3] Nivedita Rahul, "AI-Enhanced API Integrations: Advancing Guidewire Ecosystems with Real-Time Data," International Journal of Emerging Research in Engineering and Technology, 2021. Available: https://ijeret.org/index.php/ijeret/article/download/255/243

[4] Felix Neubauer et al., "AI-assisted JSON Schema Creation and Mapping," in 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2025. Available: https://arxiv.org/pdf/2508.05192

[5] Erhard Rahm and Philip A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," The VLDB Journal, 2001. Available: https://link.springer.com/article/10.1007/s007780100057

[6] Hong-Hai Do and Erhard Rahm, "COMA: A System for Flexible Combination of Schema Matching Approaches," in VLDB'02: Proceedings of the 28th International Conference on Very Large Databases, Morgan Kaufmann, 2002. Available: https://www.vldb.org/conf/2002/S17P03.pdf

[7] Hong-Hai Do and Erhard Rahm, "Matching Large Schemas: Approaches and Evaluation," Information Systems, 2007. Available: https://www.sciencedirect.com/science/article/abs/pii/S0306437906000780

[8] Ronald Fagin et al., "Clio: Schema Mapping Creation and Data Exchange," in Conceptual Modeling: Foundations and Applications, Springer Berlin Heidelberg, 2009. Available: https://link.springer.com/chapter/10.1007/978-3-642-02463-4_12

[9] A. H. Doan, A. Halevy, and Z. Ives, "The Future of Data Integration," in Principles of Data Integration, 2012. Available: https://dl.acm.org/doi/10.5555/2401764

[10] Avanika Narayan et al., "Can Foundation Models Wrangle Your Data?," Proceedings of the VLDB Endowment, 2022. Available: https://dl.acm.org/doi/10.14778/3574245.3574258

[11] Sushant Kumar et al., "Opportunities and Challenges in Data-Centric AI," IEEE Access, vol. 12, 2024. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10444552

[12] OpenAPI Initiative, "OpenAPI Specification v3.2.0," 2025. Available: https://spec.openapis.org/oas/latest.html

[13] AsyncAPI Initiative, "AsyncAPI Specification v3," 2023. Available: https://www.asyncapi.com/docs/reference/specification/v3.0.0

[14] Pact, "Integration Testing Done Properly," 2023. Available: https://pact.io/

[15] CloudEvents, "CloudEvents v1.0.2 Specification," 2022. Available: https://cloudevents.io

[16] OpenTelemetry, "Specification and Instrumentation." Available: https://opentelemetry.io

[17] Open Policy Agent (OPA), "Policy-as-code." Available: https://www.openpolicyagent.org

[18] W3C, "Trace Context Recommendation," 2012. Available: https://www.w3.org/TR/trace-context/

[19] UCUM, "Unified Code for Units of Measure." Available: https://ucum.org

[20] ISO 4217, "Currency Codes." Available: https://www.iso.org/iso-4217-currency-codes.html