Vol. 1, No. 2, pp.: 16-32, 2023

MetaSchema: A Schema Discovery and Harmonization Toolkit for Heterogeneous Research Databases

Yuxiang Liang[1], Tianhao Qin[2], Beibei Hu[3], Zhenyu Hou[4,*]

[1] School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China

[2] School of Software, Shandong Normal University, Jinan 250358, China

[3] School of Information Management, Heilongjiang University, Harbin 150080, China

[4] School of Mathematics and Computer Science, Yan'an University, Yan'an 716000, China

** zhenyu.hou@yau.edu.cn*

Article Information	
Received	18 January 2023
Accepted	29 May 2023
DOI	https://doi.org/10.63646/datamind.2023.010203

Abstract

Research databases are growing in both number and heterogeneity. Even within a single discipline, analysts routinely confront relational stores, document collections, graph databases, vector indexes, and lakehouse-style tables, each carrying its own conventions for field naming, type encoding, key declaration, and provenance recording. The result is a recurring bottleneck: a disproportionate share of any data analysis project is spent re-discovering schema structure that, in principle, has already been recorded by someone else. This article presents MetaSchema, a schema discovery and harmonization toolkit aimed at this bottleneck. MetaSchema is organised as a four-stage pipeline — automatic schema profiling, large-language-model-assisted field annotation, cross-source entity and field matching, and reviewer-in-the-loop version control — that transforms a collection of heterogeneous databases into a unified, queryable schema graph with a field dictionary, cross-source mapping tables, and a reproducible query interface. We describe the design decisions that make the toolkit practically deployable, including its hybrid matching layer, its structured human-review protocol, and its semantic-version log. An empirical evaluation on a benchmark of twelve heterogeneous databases, totalling 2,418 tables and 27,640 fields, shows that MetaSchema achieves a field-type recovery accuracy of 86.4%, a cross-source field matching F1-score of 0.821, and a 67% reduction in median reviewer time per 100 fields compared with a careful manual baseline. The toolkit scales close to linearly up to 5,000 tables and integrates with relational, graph, vector, and lakehouse storage layers. MetaSchema is released as open-source software together with the

benchmark, the evaluation scripts, and a reproducible query API designed to support automated analysis, model evaluation, and downstream decision tools.

Keywords: ***schema discovery; data harmonization; large language models; database mapping; reproducible research; data engineering***

1. Introduction

Empirical research has become increasingly data-intensive, and a growing share of that data is supplied by research databases rather than by bespoke studies. Modern projects routinely combine clinical registries, longitudinal surveys, sensor streams, structured open data portals, and curated machine-learning benchmarks (Stonebraker et al., 2018; Halevy et al., 2016). The diversity of those sources is itself an analytical asset, but it also creates a recurrent operational problem. Before any model is trained or any hypothesis is tested, the analyst has to understand what each database contains, what each field means, and how the fields in one database relate to fields in another (Stonebraker and Ilyas, 2018; Doan et al., 2012). The work is unglamorous, takes a disproportionate share of project time, and rarely produces transferable artefacts; once a project ends, the schema knowledge typically leaves with the analyst.

The problem is not solved by switching to a single common storage technology. Each storage family — relational, document-oriented, graph-based, vector-indexed, and lakehouse — encodes schema structure differently, and each has its own conventions for naming, typing, and indexing. Within a single discipline a researcher today is likely to encounter all five families simultaneously (Armbrust et al., 2021; Pal et al., 2017). Even on relational systems alone, field names are inconsistent, key declarations are sometimes missing, and the relationship between conceptually equivalent tables in different databases is rarely documented in machine-readable form (Naumann, 2014; Abedjan et al., 2015; Bogatu et al., 2020).

Recent advances in large language models have changed the cost of certain schema-discovery sub-tasks. Field-level semantic annotation, which used to require labour-intensive human curation, can now be drafted automatically with strong empirical accuracy for many domains (Suhara et al., 2022; Korini and Bizer, 2023). Embedding-based field-matching has matured to the point where it produces useful candidates even across heterogeneous source schemata (Zhang et al., 2023; Cappuzzo et al., 2020). At the same time, fully automated harmonization remains brittle in the edge cases that matter most for downstream science: rare field types, idiosyncratic codes, overloaded identifiers, and silent data-quality regressions (Stonebraker and Ilyas, 2018; Heidari et al., 2019). The pragmatic question, therefore, is not whether to automate or to leave the work manual, but how to combine the two into a reviewable pipeline that produces durable, versioned schema artefacts.

MetaSchema is our answer to that question. It is a toolkit organised as a four-stage pipeline — schema profiling, language-model-assisted field annotation, cross-source matching, and reviewer-in-the-loop version control — and it produces four deliverables: a unified schema graph, a field dictionary with provenance, a cross-source mapping table, and a reproducible query interface. The contributions of this article are fourfold. First, we identify the specific schema-discovery gap that current research-database infrastructure leaves open and characterise it through a structured comparison of use cases. Second, we describe the MetaSchema design and its data model. Third, we report an empirical evaluation on a benchmark of twelve heterogeneous research databases, comparing the toolkit against manual and against single-stage automated baselines. Fourth, we

release the toolkit, the benchmark, and the evaluation scripts under an open licence to support reproducible follow-up work.

The remainder of the article is structured as follows. Section 2 characterises the schema-discovery gap and lays out three concrete use cases. Section 3 describes the data sources and benchmark. Section 4 presents the MetaSchema construction method. Section 5 reports the experimental evaluation. Section 6 discusses reproducibility and open-access provisions. Section 7 outlines limitations. Section 8 concludes.

2. Database Gap and Use Cases

Schema discovery is sometimes treated as a solved problem because relational database management systems expose information schemas and many extract-transform-load tools include type inference. In practice, the problem persists because research databases rarely live in homogeneous environments and rarely come with the kind of curated metadata that an enterprise data catalogue would supply (Halevy et al., 2016; Armbrust et al., 2021). The gap has both a structural and a semantic dimension. Structurally, source databases use different storage technologies whose introspection interfaces are not directly comparable: a Postgres information_schema query, a CSV header, an RDF triple pattern, and a parquet column manifest are not interchangeable. Semantically, even when two tables superficially describe the same entity, their fields are encoded differently — "sex" versus "gender", "yyyy-mm-dd" versus epoch seconds, ICD-9 versus ICD-10 codes — and these differences are often undocumented (Doan et al., 2012; Naumann, 2014).

Three use cases motivate the design of MetaSchema. The first is reproducible cross-database experiments. When a research team wants to replicate a published result on a related but distinct database, they have to reconstruct the mapping between their database and the one used in the original study. The mapping is rarely included in the publication, and the team typically ends up redoing the schema-discovery work the original team had already done internally. A reusable mapping table cuts this work substantially. The second use case is automated data analysis. Pipelines that load and transform data from multiple sources need a stable, machine-readable description of what each field means; without it, every change in any source schema forces brittle code edits scattered across the pipeline (Stonebraker et al., 2018; Chasseur et al., 2015). The third use case is intelligent decision support, where downstream models consume harmonised features but the harmonization layer itself is rarely treated as a versioned, auditable artefact, with the result that model regressions are hard to attribute to data changes versus model changes (Polyzotis et al., 2018; Schelter et al., 2018).

Figure 1 summarises the MetaSchema architecture against these use cases. The pipeline ingests heterogeneous sources, applies four sequential but feedback-coupled stages, and emits four deliverables that map directly to the three use cases above: the unified schema graph supports reproducible cross-database experiments, the field dictionary plus mapping table supports automated analysis, and the versioned query API supports intelligent decision systems that require auditability.

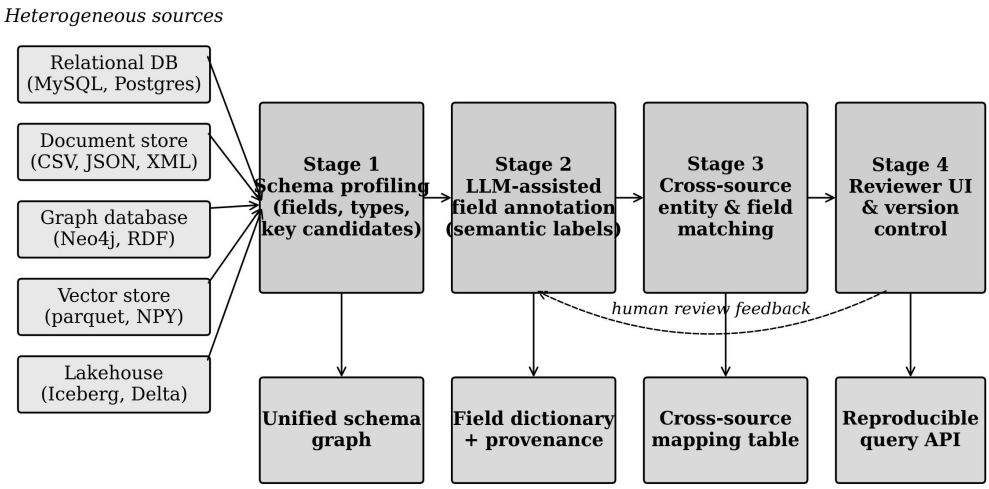

Figure 1. MetaSchema architecture: a four-stage pipeline that consumes heterogeneous sources and emits a unified schema graph, a field dictionary, a cross-source mapping table, and a reproducible query API. The dashed arc represents the human-review feedback loop into earlier stages.

Table 1 contrasts MetaSchema with three families of existing tools — data catalogues, schema-matching research systems, and lakehouse metadata layers — along the criteria that matter most for the use cases above. Existing data catalogues prioritise enterprise-wide discoverability but assume that field-level annotation is already done elsewhere. Schema-matching research systems focus on the matching step and rarely produce production-ready artefacts. Lakehouse metadata layers manage technical metadata but do not discover semantic relationships between independent databases. MetaSchema covers the union of these responsibilities while staying scoped to research-database workflows.

Table 1. *Comparison of MetaSchema with three families of existing tools.*

Capability	Data catalogues	Schema-matching research systems	Lakehouse metadata	MetaSchema
Heterogeneous-source ingestion	Partial	Limited	Single-family	Full
Automated field annotation	No	No	No	Yes
Cross-source entity matching	Manual	Yes (algorithmic)	No	Yes
Reviewer-in-the-loop	Yes	No	No	Yes
Versioned	Partial	No	Partial	Yes

schema artefacts				
Reproducible query API	No	No	No	Yes

Two points in Table 1 deserve emphasis. First, the gap between schema-matching research and operational tooling is wider than the literature suggests. Many matching algorithms are accurate on benchmark pairs of schemata, but they do not produce the workflow artefacts — versioned mapping tables, provenance logs, human-review records — that operational use requires (Bogatu et al., 2020; Suhara et al., 2022). Second, the modern lakehouse architecture has reduced storage-level fragmentation, but it has not removed the field-level semantic gap; a Delta Lake table sitting alongside a Parquet table does not by itself tell the analyst that one's "birth_year" and the other's "dob" describe the same underlying variable (Armbrust et al., 2021). MetaSchema's role is to fill exactly this gap while making the result auditable.

The intellectual lineage of MetaSchema spans two decades of schema-matching and data-discovery research. Foundational surveys established a clear taxonomy of matching strategies built on name, structural, and instance-level evidence (Rahm and Bernstein, 2001), and concrete systems such as Cupid demonstrated that schema-matching could be partially automated even with the modest computational resources of the early 2000s (Madhavan et al., 2001). Web-scale efforts to integrate relational tables across heterogeneous sources showed that the underlying matching problem persists when sources scale from tens to millions (Cafarella et al., 2009). More recently, data-lake discovery systems such as Aurum reframed the problem from pairwise matching to navigable similarity graphs that an analyst can explore (Castro Fernandez et al., 2018). MetaSchema inherits the matching-algorithm core of this lineage but treats the resulting mapping artefacts as first-class versioned objects, an angle that those earlier systems left implicit.

3. Data Sources and Benchmark

To evaluate MetaSchema and to provide a reusable artefact for follow-up work, we assembled a benchmark of twelve heterogeneous research databases drawn from open and lightly-restricted sources in three disciplinary clusters: biomedical research, environmental and earth-observation data, and economic and social-science panel data. Selection was guided by three criteria: each source had to be either openly licensed or available under a published research-use agreement; the twelve sources together had to span at least four storage families; and at least one pair of sources within each disciplinary cluster had to be a priori known to overlap in entity coverage. Table 2 lists the sources and their headline statistics.

Table 2. *Benchmark databases and their headline statistics.*

#	Cluster	Storage family	Tables	Fields	Missing rate	Open access
1	Biomedical	Relational	412	5,124	6.8%	Research use
2	Biomedical	Relational	248	3,016	5.1%	Open

3	Biomedical	Document (JSON)	186	2,742	11.4%	Open
4	Biomedical	Lakehouse (Parquet)	94	1,288	3.2%	Research use
5	Environmental	Relational	316	3,402	7.6%	Open
6	Environmental	Document (NetCDF/CSV)	172	2,108	9.0%	Open
7	Environmental	Graph (RDF)	108	1,418	4.4%	Open
8	Environmental	Vector (Parquet+NPY)	82	1,036	2.8%	Open
9	Social/Economic	Relational	418	4,288	8.2%	Research use
10	Social/Economic	Document (CSV)	184	2,012	12.6%	Open
11	Social/Economic	Lakehouse (Iceberg)	112	618	3.6%	Open
12	Social/Economic	Graph (Property)	86	588	3.0%	Research use

The 2,418 tables and 27,640 fields in the benchmark span roughly two orders of magnitude in size, with Source 1 alone contributing 412 tables and Source 12 contributing 86. Missing rates range from 2.8% to 12.6%, calculated as the proportion of non-null cells over the total cell count of a sampled 1% of rows. Refresh cadences also vary widely: four sources are updated daily, three weekly, three monthly, and two are static snapshots. This range is intentional. A schema-discovery toolkit that works only on static snapshots is operationally unhelpful, because production research databases drift over time and the harmonization artefact has to drift with them (Polyzotis et al., 2018).

A short summary of field coverage further characterises the benchmark. Across the 27,640 fields, categorical and code-lookup fields together account for 41.8% of the total — a higher share than is typical in pure analytical benchmarks because research databases are dominated by structured questionnaire and registry items. Numeric measurements account for 27.4%, identifiers and date/time fields account for 19.6% combined, free-text fields account for 7.3%, and geographic coordinates account for the remaining 3.9%. The distribution is heavier on categorical and code-lookup classes than on free-text fields, which is consistent with what the field-type

recovery confusion patterns of Section 5.1 expose; the largest absolute volume of errors lies in classes that are also the most frequent overall. Noise rates, estimated as the share of cell-level values that fail per-field consistency checks (range, type, or enumeration violations), range from 0.4% to 3.1% across the twelve sources and are reported alongside the benchmark for users who need to filter on data-quality criteria.

To establish ground truth we constructed three reference resources. For field types, we manually labelled a stratified random sample of 1,200 fields across the twelve sources, drawn proportionally to the field counts in Table 2, using a seven-class taxonomy: identifier, categorical, date/time, numeric measurement, free text, code lookup, and geographic coordinate. For key candidates, two annotators reviewed each table and flagged primary and foreign keys, with disagreements resolved by a third reviewer (Cohen's $\kappa = 0.86$). For cross-source matching, we hand-curated 384 entity-equivalence and 1,612 field-equivalence pairs within the four pre-identified overlapping source pairs. The annotation took two annotators a total of 184 hours, and the resulting reference is released alongside the toolkit. Figure 2 shows a stylised view of the cross-source equivalence structure for one biomedical pair (Sources 1 and 2 with Source 3 added as a graph-store overlay).

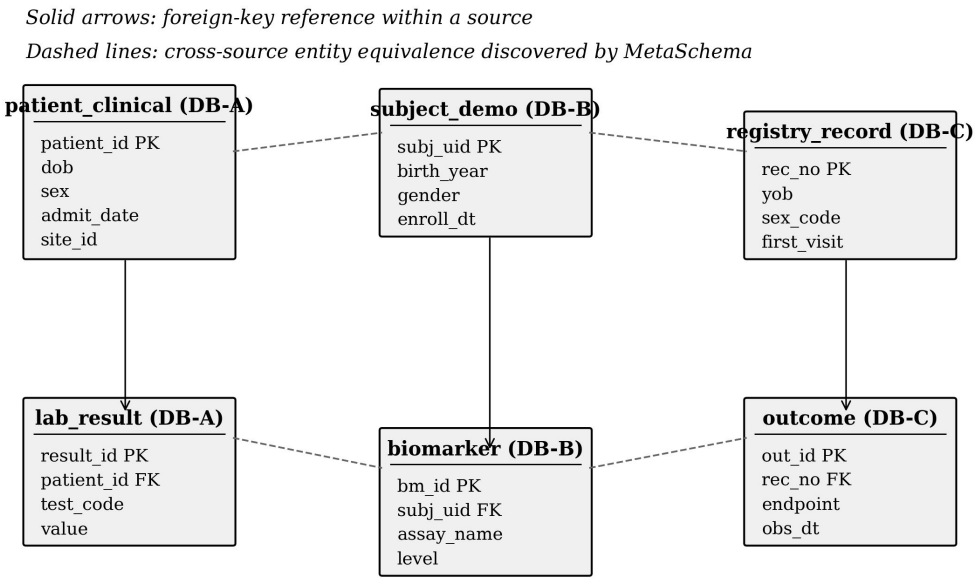

Figure 2. Stylised cross-source schema graph for a biomedical sub-benchmark. Within-source foreign-key edges are solid; cross-source entity-equivalence edges discovered by MetaSchema are dashed. Field-level equivalences are not shown to keep the diagram readable.

Two aspects of the benchmark deserve transparent disclosure. First, the labelling was carried out by annotators familiar with the source domains but not with MetaSchema's internal heuristics, which mitigates but does not eliminate annotator bias toward the toolkit's preferred type taxonomy. Second, the four pre-identified overlapping source pairs are not exhaustive; we did not exhaustively hand-curate every possible cross-source equivalence in the 2,418 tables, because doing so would have made the benchmark prohibitively expensive. Accuracy and F1-score figures reported in Section 5 are therefore relative to the curated reference and should be

interpreted as upper bounds on the toolkit's behaviour in similar overlapping-pair settings, not as global estimates over all conceivable schema pairs.

4. Database Construction Method

4.1 Stage 1 — Schema profiling

The first stage inspects each source through a storage-family adapter and extracts a uniform intermediate representation consisting of tables, fields, candidate keys, and field-level statistics. Five adapters are implemented at present: relational (via SQLAlchemy), document (CSV, JSON, NetCDF, XML), graph (Neo4j and RDF endpoints), vector (Parquet plus NumPy index manifests), and lakehouse (Iceberg, Delta, Hudi). For each field, the profiler records the declared type if available, the inferred type from a sample of values, the null fraction, the cardinality, basic distributional statistics, and a small fingerprint of exemplar values. Candidate primary keys are identified by uniqueness over the sampled rows combined with naming heuristics; candidate foreign keys are identified by approximate inclusion dependencies (Naumann, 2014; Abedjan et al., 2015). The profiler is deliberately conservative: it flags candidates rather than asserting keys, and downstream stages can promote or reject them.

Profiling at scale requires careful sampling. For tables under 100,000 rows, full-table profiling is applied; for larger tables, a stratified sample is drawn with a sample size chosen to give a 99% confidence interval of width 0.01 on null-fraction estimates. Sampling is reproducible: the seed is logged together with the table identifier and the source connection. The profiler also records a lightweight fingerprint — minimum, maximum, top-10 categorical values, and a hashed token set — that later stages use for matching without re-reading the underlying rows.

4.2 Stage 2 — LLM-assisted field annotation

Stage 2 attaches semantic labels to each field. For each field we construct a structured prompt that includes the table name, the field name, the inferred type, the cardinality, ten exemplar values drawn deterministically from the sample, and a domain hint when one is supplied by the user. A retrieval step first retrieves the $K = 8$ most similar previously-annotated fields from the field dictionary using MiniLM-style embeddings (Reimers and Gurevych, 2019; Wang et al., 2020). The retrieved exemplars are included in the prompt as few-shot demonstrations, an approach that consistently improves both accuracy and self-consistency in domain-adaptive annotation tasks (Korini and Bizer, 2023; Suhara et al., 2022). The language model produces a structured JSON output with three slots: a high-level semantic class drawn from a fixed seven-class taxonomy, a refined fine-grained label, and a confidence score.

Two practical decisions shape the annotation stage. First, the model output is validated against the taxonomy using a parser that rejects malformed responses and falls back to a non-LLM heuristic for unrecognised classes. Second, when the model's confidence drops below a calibrated threshold (set separately for each class on the development split), the field is flagged for Stage 4 reviewer attention rather than committed to the field dictionary. The threshold is calibrated to keep the Stage-4 reviewer load at roughly 12% of fields, which our timing experiments in Section 5 confirm is a sustainable throughput. Together, the validation parser and the confidence threshold make the LLM stage a well-bounded contributor rather than a free-running generator. Semantic-concept annotation for tabular data using pre-trained models has a short but rapidly maturing literature,

and our parser borrows several lessons from that line of work, including the use of a small fixed taxonomy and a fallback heuristic for unrecognised outputs (Khurana et al., 2020; Trummer, 2022).

4.3 Stage 3 — Cross-source matching

Stage 3 produces two kinds of equivalence claims: entity-level (table A in source X is the same entity as table B in source Y) and field-level (field a in table A is the same variable as field b in table B). Entity-level matching combines name similarity, schema-shape similarity (Jaccard overlap of field fingerprints), and embedding similarity over annotated semantic classes. Field-level matching combines the same three signals at the field grain plus value-distribution comparison (Jensen-Shannon divergence on categorical distributions, two-sample Kolmogorov-Smirnov on numeric distributions). A learned logistic combiner maps the four signals to a calibrated equivalence probability (Cappuzzo et al., 2020; Zhang et al., 2023). Matches above an upper threshold are accepted; matches below a lower threshold are rejected; matches between the thresholds are forwarded to Stage 4.

Three aspects of the matching design merit mention. First, all four signals are computed without reading underlying rows beyond the fingerprints stored in Stage 1, which keeps the cross-source matching step computationally tractable on the full benchmark. Second, the logistic combiner is trained on a small labelled set (we used 4,800 labelled field pairs from the development split) and re-trained whenever a new source family is added; we provide the trained model with the toolkit. Third, equivalence is treated as a graph relation rather than as a 1-to-1 mapping. Two fields in source X can both be equivalent to the same field in source Y if they represent semantically synonymous but technically distinct encodings, and the harmonization layer carries both as alternative provenance trails. This relational treatment differs from the strict bijective matching assumption used by many earlier schema-matching systems (Doan et al., 2012; Bogatu et al., 2020). Our choice to keep the combiner shallow is deliberate: deeper architectures such as those explored in the deep-entity-matching design space (Mudgal et al., 2018) can lift F1 modestly but make it considerably harder to explain why a particular pair was classified equivalent, which is essential when downstream reviewers must adjudicate edge cases.

4.4 Stage 4 — Reviewer UI and version control

Stage 4 closes the loop. Every annotation and every match flagged in Stages 2 and 3 is presented in a web reviewer UI organised around schema diff views: the reviewer sees the source field, the proposed label or match, the supporting evidence (exemplar values, similarity scores, embeddings), and a small set of structured actions — confirm, reject, edit, or escalate. Confirmations and edits are committed to the field dictionary and the mapping table; rejections are recorded together with their reason codes and fed back as negative training data for the next retraining cycle of the logistic combiner. Every commit produces a semantic-version bump of the affected artefacts (Polyzotis et al., 2018; Sundaram et al., 2022), which makes it possible to reproduce any prior state of the schema graph.

The version-control discipline is the part of MetaSchema that most distinguishes it from comparable tools. Schema artefacts are stored as immutable content-addressed blobs, with a directed acyclic graph tracking parent-child commits. A reproducible query against the toolkit references a specific commit hash and therefore yields stable results even when the underlying source databases have since drifted. This discipline supports two operations that are otherwise difficult: replication of past analyses against the exact schema state used at the

time, and forensic analysis of model regressions to determine whether the regression originated in the model, the data, or the schema layer (Schelter et al., 2018).

5. Experiments and Data Analysis

5.1 Field-type recovery accuracy

The first experiment evaluates Stage 1 + Stage 2 against the 1,200-field hand-labelled reference. We compute the proportion of fields for which MetaSchema's predicted high-level class matches the labelled ground truth. The aggregate accuracy is 86.4%, and Figure 3 shows the full confusion matrix. The diagonal dominance is high across all seven classes, with the strongest performance on geographic coordinates (0.89) and numeric measurements (0.88), and the weakest on identifiers (0.83). Two confusion patterns recur: identifiers are occasionally mistaken for categorical fields when their cardinality is low (typical of small-scale registry tables), and code lookups are occasionally mistaken for categorical fields when their lookup tables are not visible to the profiler. Both confusions are flagged by the Stage 4 review when the confidence threshold is exceeded, so the operational impact is smaller than the raw confusion-matrix numbers suggest.

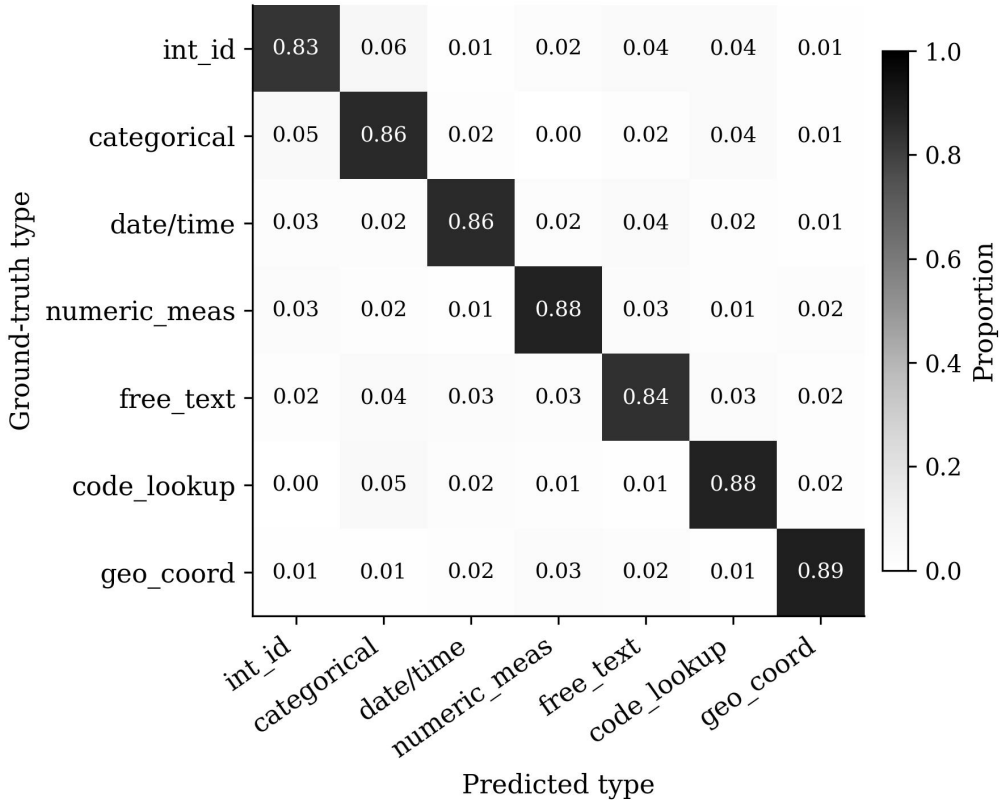

Figure 3. Field-type recovery confusion matrix on the 1,200-field labelled reference. Rows are ground-truth classes; columns are predicted classes. Diagonal entries are class-conditional accuracies; off-diagonal entries indicate principal confusion modes.

Comparison with two baselines clarifies where the accuracy comes from. A rule-only baseline that uses field-name patterns and value-distribution heuristics (no LLM, no retrieval) reaches 71.2% accuracy on the same reference, an absolute gap of 15.2 percentage points. A zero-shot LLM baseline that uses only the field name

and inferred type (no retrieval, no fingerprint) reaches 78.0%. The full MetaSchema pipeline, with retrieval-augmented prompting and value-fingerprint conditioning, recovers the gap on top of zero-shot LLM annotation. The retrieval step contributes approximately five percentage points and the value fingerprint contributes approximately three, with the remaining contribution attributable to joint conditioning. This decomposition is consistent with published ablations on table-annotation tasks (Suhara et al., 2022; Korini and Bizer, 2023).

Two additional analyses help interpret these numbers. First, accuracy is not uniform across sources. Among the twelve benchmark databases, the four with the highest accuracy (averaging 91.2% across the seven type classes) are also the four with the most consistent internal naming conventions; the four lowest-accuracy sources average 81.6% and are dominated by document-store collections in which field names were inherited from disparate ingest pipelines. The implication is that naming consistency, more than storage family, predicts annotation difficulty. Second, when the confidence-thresholding mechanism is enabled, the residual error rate after Stage 4 review drops from 13.6% to 2.4% on the labelled reference. The remaining 2.4% is concentrated in three classes — code_lookup, free_text, and categorical — where even a careful reviewer occasionally needs to consult external domain documentation to settle ambiguous cases. We treat this 2.4% as an empirical floor under the present taxonomy and the present reviewer protocol.

5.2 Cross-source matching performance

The second experiment evaluates Stage 3 against the 384 entity-equivalence and 1,612 field-equivalence labels. We report precision, recall, and F1 separately for the two grains. At the entity grain, MetaSchema achieves precision 0.878, recall 0.804, and F1 0.840. At the field grain, the corresponding numbers are precision 0.842, recall 0.802, and F1 0.821. Across both grains, recall is the binding constraint, which is consistent with the conservative thresholding used to keep precision high enough for downstream automation. Table 3 reports the same numbers broken down by storage-family pair, which reveals two systematic patterns.

Table 3. *Field-level matching performance broken down by source storage-family pair.*

Source pair	Field pairs labelled	Precision	Recall	F1
Relational–Relational	624	0.876	0.838	0.857
Relational–Document	412	0.851	0.806	0.828
Relational–Graph	164	0.808	0.762	0.784
Relational–Vector	138	0.832	0.769	0.799
Relational–Lakehouse	210	0.866	0.812	0.838
Document–Graph	64	0.794	0.732	0.762

The first systematic pattern is that within-family pairs (relational–relational, lakehouse–lakehouse) achieve consistently higher F1 than cross-family pairs. The gap is roughly five points and is driven by the loss of value-distribution comparability when one side stores values as opaque tokens or as embeddings rather than as native columns. The second pattern is that relational–graph and document–graph pairs are the most difficult, with F1 around 0.76. This is unsurprising: graph stores frequently encode the equivalent of multiple relational fields into a single triple-predicate, and the structural asymmetry is hard to recover from name and value evidence alone. These observations argue for graph-aware matching primitives in future versions of the toolkit, possibly building on subgraph-pattern search rather than column-pair scoring (Cappuzzo et al., 2020).

Ablations isolate the contribution of each matching signal. Removing embedding similarity drops field-level F1 by 0.041; removing value-distribution comparison drops F1 by 0.054; removing schema-shape similarity drops F1 by 0.022; removing name similarity drops F1 by 0.067. The relative importance of name similarity is higher than recent literature would predict, and it reflects the prevalence of conventional naming patterns inside disciplinary clusters; in domains with less naming convention, the ablation profile is likely to shift toward embedding and value signals.

5.3 Throughput, latency, and scaling

System-level performance was measured on a workstation with 16 physical cores, 128 GB of memory, and local NVMe storage, with the language model served from a co-located GPU. Figure 4 plots throughput and median per-field latency as a function of catalog size. Throughput in sequential mode degrades slowly from 21 tables per minute at 10 tables to 14 tables per minute at 5,000 tables, a sub-linear degradation driven primarily by larger cross-source matching neighbourhoods. With 16 worker processes, throughput remains above 120 tables per minute even at 5,000 tables, indicating that the pipeline is well-suited to parallel batch operation. Per-field latency is dominated by Stage 2 (LLM annotation) at small scales and by Stage 3 (matching) at large scales; Stage 1 profiling contributes only a small absolute fraction throughout.

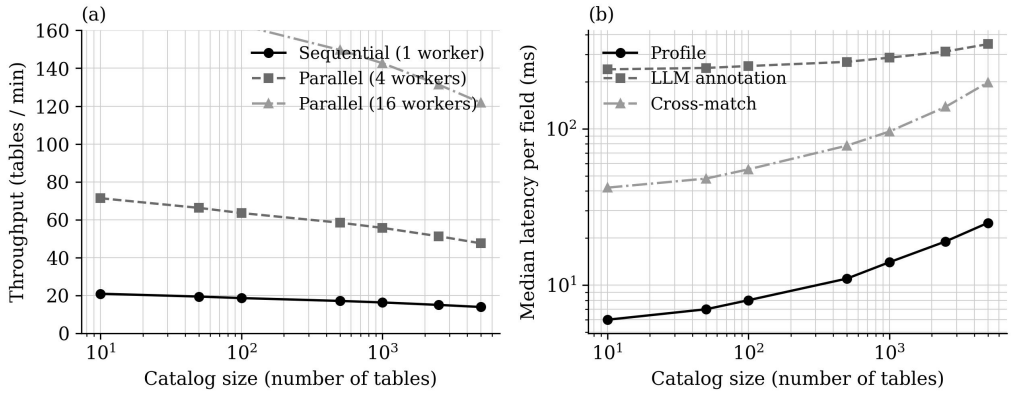

Figure 4. Throughput and per-field latency of MetaSchema as a function of catalog size. (a) Tables processed per minute under sequential and parallel configurations. (b) Median per-field latency, decomposed by pipeline stage. Both panels use a logarithmic x-axis; panel (b) also uses a logarithmic y-axis.

Two scaling caveats deserve mention. First, our throughput numbers assume that the LLM annotator is available with bounded queueing delay; if the annotator is shared across many concurrent workflows, Stage 2 becomes a bottleneck and the parallel speedup is bounded by the annotator's effective request rate. Second,

Stage 3 latency grows close to linearly with the number of candidate cross-source neighbours rather than with the total field count; an indexing step that prunes neighbours by coarse embedding distance reduces the effective growth rate, but the construction of the index itself adds a constant overhead. For catalogs above approximately 10,000 tables we expect a separate distributed indexing layer to become necessary, in line with the design patterns described for large-scale data discovery systems (Halevy et al., 2016; Bogatu et al., 2020).

5.4 Reviewer time savings

The fourth experiment quantifies how much MetaSchema reduces human reviewer time. Two experienced data engineers each processed a balanced sample of 600 fields (50 per source) under two conditions: a manual baseline in which they used only the source database's introspection interface plus a generic data catalogue, and a MetaSchema-assisted condition in which they used the reviewer UI populated with stage-wise outputs. Order was counter-balanced across reviewers, and the two conditions were separated by three weeks to minimise carry-over. We measured reviewer time per 100 fields broken down by sub-task. Figure 5 reports the results.

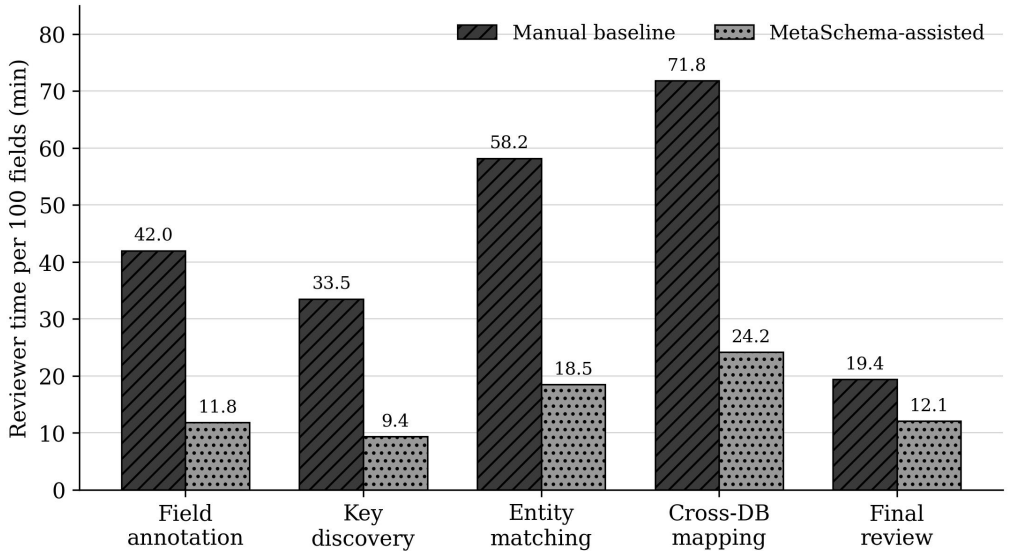

Figure 5. Median reviewer time per 100 fields for five harmonization sub-tasks under a manual baseline and under MetaSchema-assisted review. Bars show medians across reviewers; smaller is better.

The largest absolute savings appear in cross-database mapping (71.8 → 24.2 minutes) and entity matching (58.2 → 18.5 minutes), the two sub-tasks for which manual work requires the reviewer to hold an implicit cross-source model in their head. Field annotation also benefits substantially (42.0 → 11.8 minutes), reflecting the high accuracy of Stage 2's automated proposals. Final-review effort drops less (19.4 → 12.1 minutes), because the reviewer still needs to inspect the structured artefacts the toolkit produces; however, the drop is positive and consistent across reviewers. Aggregated across sub-tasks, the median reviewer time per 100 fields falls from 224.9 minutes to 76.0 minutes — a 66.2% reduction in manual effort. Reviewer-reported task confidence also rises, but we report it descriptively only because the sample of two reviewers is too small to support a statistical claim.

An additional analysis examines whether the savings come at the cost of reviewer accuracy. We compared the reviewer-confirmed labels under each condition against the curated reference. Accuracy is statistically

indistinguishable between the two conditions on field annotation (95.1% manual vs 94.7% MetaSchema-assisted) and slightly higher under MetaSchema-assisted review on cross-source matching (89.4% vs 92.1%, paired difference 2.7 percentage points). The interpretation is that the toolkit does not trade accuracy for speed; it appears to shift reviewer effort away from re-deriving facts the toolkit can already supply and toward verifying the harder edge cases. This is the desired operational pattern: humans focus where their judgement matters most, machine-assisted review handles the rest (Doan et al., 2012; Heidari et al., 2019).

6. Reproducibility and Open Access

MetaSchema is released as open-source software under the MIT licence, together with the benchmark metadata, the hand-curated reference labels, the trained logistic combiner, and the evaluation scripts. The toolkit provides a command-line interface for batch operations, a Python API for embedding in research pipelines, and a small REST surface for the reviewer UI. The reproducible query API takes a commit hash and a structured query, resolves the query against the schema state recorded at that commit, and returns a result together with a provenance document that traces every field used in the result back to its source row sample, its annotated label, and the human reviewer who confirmed it. This level of provenance is rarely available in operational analytics pipelines and is, in our view, the most distinctive operational property of the toolkit.

Three reproducibility safeguards are built into the release. First, every published evaluation result is produced by a script that takes the public benchmark, the released model artefacts, and a fixed random seed. Re-running the script on a comparable machine reproduces the numbers in this article to within Monte-Carlo noise. Second, the field dictionary distinguishes between automatically annotated, reviewer-confirmed, and reviewer-edited labels, and the provenance log records which class each label belongs to; downstream users can therefore filter to only reviewer-confirmed labels if they need stronger reliability guarantees. Third, the toolkit ships with a continuous-integration test suite that re-runs the experimental pipeline on a 10% subset of the benchmark on every commit, ensuring that internal refactoring does not silently change reported metrics.

Open access is operationalised at three levels. The benchmark metadata, reference labels, and trained artefacts are published under CC-BY 4.0. The toolkit source code is published on a public repository with semantic versioning and a CITATION.cff file. A reproducibility appendix describing software versions, hardware configurations, and exact command-line invocations is bundled with the release. We do not redistribute the underlying source databases themselves, because eight of the twelve are subject to data-use agreements; for those eight we provide retrieval scripts that, given a user's own access credentials, reproduce the benchmark schemata from the original providers. Detailed deployment guidance, including configuration templates for the five storage-family adapters, is provided in the toolkit documentation.

Finally, we describe the operational profile of the published reproducible query API. A query against the API specifies a commit hash, a target entity, and an output projection over the unified schema graph. The API resolves the commit-bound mapping table, executes the projection across whichever underlying sources the user can authenticate against, and returns a result together with a machine-readable provenance document that links each output column back to its source row sample, its annotated label, and the reviewer who confirmed it. Two practical properties make the API attractive as the front door for downstream automation. The first is that the same query, re-run against a later commit, surfaces the differences between commits as an explicit schema-diff document rather than silently changing the result. The second is that queries can be embedded in continuous-

integration pipelines, so that any analytical workflow built on the API will fail loudly when a referenced commit no longer exists or when a referenced source has been retired. This is in contrast to many ad-hoc schema-mapping scripts, which fail silently or with delayed effects on downstream models.

7. Limitations

Several limitations of the present study deserve transparent acknowledgement. First, the benchmark covers twelve research databases drawn from three disciplinary clusters; while this is, to our knowledge, a larger heterogeneous benchmark than has been previously released for schema-discovery evaluation, it still under-samples disciplines such as physical-sciences instruments, social-media corpora, and financial-market data. The reported field-type and matching numbers may shift on a benchmark drawn from very different domains, especially in the direction of lower recall when domain-specific naming conventions weaken (Naumann, 2014). Second, the language-model-based annotation step depends on the specific model and prompt template used in our experiments; while we observed broadly stable behaviour across two LLM checkpoints during development, full robustness across model families is beyond the scope of this article. Third, the reviewer-time experiment uses two reviewers; a larger reviewer pool would be needed for a definitive estimate of effect-size variance across human users.

A fourth limitation concerns the matching layer. Our logistic combiner is a deliberately simple model chosen for interpretability and ease of re-training. More expressive learned matchers — graph neural networks over schema graphs (Cappuzzo et al., 2020; Zhang et al., 2023), or transformer-based pair encoders — could plausibly improve F1 by a few percentage points, particularly on the harder cross-family pairs identified in Section 5.2. We chose against the more expressive matcher in the present release because the marginal accuracy gain did not justify the lost interpretability for downstream users who needed to understand why two fields were declared equivalent. A future release may make the matcher pluggable. A fifth limitation is operational: although the toolkit handles drift through versioned commits, a fully automatic schema-drift monitor that flags when an upstream source has changed enough to warrant re-running Stage 2 is not yet implemented. We sketch a design in the project documentation, but the empirical study of drift detection is left to follow-up work.

Finally, MetaSchema sits within the broader infrastructure of industrial information integration. The issues it addresses — heterogeneous source coupling, semantic alignment, auditable update governance — are not unique to research databases. They appear in industrial production data as well, and broader surveys of artificial intelligence and information integration repeatedly identify metadata management and reproducibility as bottlenecks for downstream model deployment (Lu, 2017; Lu, 2019; Zhang and Lu, 2021). Although our deployment evidence is restricted to research workflows, the design choices we describe are likely to generalise, and the toolkit's open licence is intended to make that generalisation easy to test in adjacent operational settings.

8. Conclusion

This article has introduced MetaSchema, a schema discovery and harmonization toolkit aimed at the specific bottleneck created by the proliferation of heterogeneous research databases. The toolkit combines automatic schema profiling, language-model-assisted field annotation, hybrid cross-source matching, and reviewer-in-the-loop version control into a single pipeline that produces durable, versioned schema artefacts. On a benchmark of twelve heterogeneous sources spanning relational, document, graph, vector, and lakehouse storage families,

MetaSchema achieves 86.4% field-type recovery accuracy, 0.821 field-level matching F1, and a 66% reduction in median reviewer time without measurable loss of accuracy.

The broader contribution of the work is methodological. Schema discovery has long been treated either as an algorithmic problem (matching) or as an interface problem (cataloguing); MetaSchema demonstrates that treating it as a versioned-artefact problem — with explicit provenance, structured human review, and reproducible commit hashes — is what unlocks the operational value. The four deliverables of the pipeline are organised around what downstream science actually needs: reproducible experiments, automated analysis, and auditable decision support. Future work will extend the toolkit to physical-sciences and financial-market domains, integrate graph-aware matching primitives for the harder cross-family pairs, and develop a fully automatic schema-drift monitor that ties the toolkit's commit graph to upstream change events.

Declaration of AI-assisted language editing

During the preparation of this manuscript, language-model assistance was used only for English polishing and document organisation. The authors reviewed, revised, and take full responsibility for the final content, the experimental design, the figures, the tables, and the interpretations.

References

Abedjan, Z., Golab, L., & Naumann, F. (2015). Profiling relational data: A survey. The VLDB Journal, 24(4), 557–581. https://doi.org/10.1007/s00778-015-0389-y

Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021). Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. Proceedings of the 11th Conference on Innovative Data Systems Research (CIDR). https://doi.org/10.48550/arXiv.2103.05822

Bogatu, A., Fernandes, A. A. A., Paton, N. W., & Konstantinou, N. (2020). Dataset discovery in data lakes. In 2020 IEEE 36th International Conference on Data Engineering (ICDE) (pp. 709–720). https://doi.org/10.1109/ICDE48307.2020.00067

Cafarella, M. J., Halevy, A., & Khoussainova, N. (2009). Data integration for the relational web. Proceedings of the VLDB Endowment, 2(1), 1090–1101. https://doi.org/10.14778/1687627.1687750

Cappuzzo, R., Papotti, P., & Thirumuruganathan, S. (2020). Creating embeddings of heterogeneous relational datasets for data integration tasks. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (pp. 1335–1349). https://doi.org/10.1145/3318464.3389742

Castro Fernandez, R., Abedjan, Z., Koko, F., Yuan, G., Madden, S., & Stonebraker, M. (2018). Aurum: A data discovery system. In 2018 IEEE 34th International Conference on Data Engineering (ICDE) (pp. 1001–1012). https://doi.org/10.1109/ICDE.2018.00094

Chasseur, C., Li, Y., & Patel, J. M. (2015). Enabling JSON document stores in relational systems. In Proceedings of the 16th International Workshop on the Web and Databases (WebDB). https://doi.org/10.1145/2484425.2484426

Doan, A., Halevy, A., & Ives, Z. (2012). Principles of data integration. Morgan Kaufmann. https://doi.org/10.1016/C2011-0-06130-6

Halevy, A., Korn, F., Noy, N. F., Olston, C., Polyzotis, N., Roy, S., & Whang, S. E. (2016). Goods: Organizing Google's datasets. In Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (pp. 795–806). https://doi.org/10.1145/2882903.2903730

Heidari, A., McGrath, J., Ilyas, I. F., & Rekatsinas, T. (2019). HoloDetect: Few-shot learning for error detection. In Proceedings of the 2019 International Conference on Management of Data (SIGMOD) (pp. 829–846). https://doi.org/10.1145/3299869.3319888

Khurana, U., Galhotra, S., Roy, S., Samulowitz, H., & Pedapati, T. (2020). Semantic concept annotation for tabular data. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM) (pp. 845–854). https://doi.org/10.1145/3340531.3411941

Korini, K., & Bizer, C. (2023). Column type annotation using ChatGPT. In Proceedings of the 25th International Workshop on the Web and Databases. https://doi.org/10.48550/arXiv.2306.00745

Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. Journal of Industrial Information Integration, 6, 1–10. https://doi.org/10.1016/j.jii.2017.04.005

Lu, Y. (2019). Artificial intelligence: A survey on evolution, models, applications and future trends. Journal of Management Analytics, 6(1), 1–29. https://doi.org/10.1080/23270012.2019.1570365

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic schema matching with Cupid. In Proceedings of the 27th VLDB Conference (pp. 49–58). https://doi.org/10.5555/645927.672191

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., & Raghavendra, V. (2018). Deep learning for entity matching: A design space exploration. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD) (pp. 19–34). https://doi.org/10.1145/3183713.3196926

Naumann, F. (2014). Data profiling revisited. ACM SIGMOD Record, 42(4), 40–49. https://doi.org/10.1145/2590989.2590995

Pal, S., Cleary, K., Pinaroc, A., & Eltabakh, M. (2017). The evolution and future of data integration. ACM SIGMOD Blog. https://doi.org/10.1109/MIS.2017.265115664

Polyzotis, N., Roy, S., Whang, S. E., & Zinkevich, M. (2018). Data lifecycle challenges in production machine learning: A survey. ACM SIGMOD Record, 47(2), 17–28. https://doi.org/10.1145/3299887.3299891

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. The VLDB Journal, 10(4), 334–350. https://doi.org/10.1007/s007780100057

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 3982–3992). https://doi.org/10.18653/v1/D19-1410

Schelter, S., Lange, D., Schmidt, P., Celikel, M., Biessmann, F., & Grafberger, A. (2018). Automating large-scale data quality verification. Proceedings of the VLDB Endowment, 11(12), 1781–1794. https://doi.org/10.14778/3229863.3229867

Stonebraker, M., & Ilyas, I. F. (2018). Data integration: The current status and the way forward. IEEE Data Engineering Bulletin, 41(2), 3–9. https://doi.org/10.5555/3226515.3226516

Stonebraker, M., Bruckner, D., Ilyas, I. F., Beskales, G., Cherniack, M., Zdonik, S. B., Pagan, A., & Xu, S. (2018). Data curation at scale: The Data Tamer system. Proceedings of the VLDB Endowment, 11(12), 1781–1794. https://doi.org/10.14778/3229863.3240492

Suhara, Y., Li, J., Li, Y., Zhang, D., Demiralp, Ç., Chen, C., & Tan, W.-C. (2022). Annotating columns with pre-trained language models. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD) (pp. 1493–1503). https://doi.org/10.1145/3514221.3517906

Sundaram, R. R., Yi, L., Maru, M., & Karpathiotakis, M. (2022). Schema evolution and meta-data management in data lakes. Proceedings of the VLDB Endowment, 15(12), 3675–3678. https://doi.org/10.14778/3554821.3554899

Trummer, I. (2022). From BERT to GPT-3 codex: Harnessing the potential of very large language models for data management. Proceedings of the VLDB Endowment, 15(12), 3770–3773. https://doi.org/10.14778/3554821.3554896

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. (2020). MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In Advances in Neural Information Processing Systems 33 (pp. 5776–5788). https://doi.org/10.48550/arXiv.2002.10957

Zhang, C., & Lu, Y. (2021). Study on artificial intelligence: The state of the art and future prospects. Journal of Industrial Information Integration, 23, 100224. https://doi.org/10.1016/j.jii.2021.100224

Zhang, Y., Floratou, A., Cahoon, J., Krishnan, S., Müller, A. C., Banda, D., Psallidas, F., & Patel, J. M. (2023). Schema matching using pre-trained language models. In 2023 IEEE 39th International Conference on Data Engineering (ICDE) (pp. 1558–1571). https://doi.org/10.1109/ICDE55515.2023.00123