Semantic Heterogeneity in the Age of Foundation Models

Benchmarks and Systems for Opaque Enterprise Schemas

Moe Kayali

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2026

Reading Committee:

Dan Suciu, Chair
Magdalena Balazinska
Leilani Battle

Program Authorized to Offer Degree:

Computer Science & Engineering

University of Washington

Abstract

Semantic Heterogeneity in the Age of Foundation Models:
Benchmarks and Systems for Opaque Enterprise Schemas

Moe Kayali

Chair of the Supervisory Committee:
Dan Suciu
Computer Science & Engineering

Semantic heterogeneity—the fact that the same real-world concept is represented differently across databases—is the principal obstacle to integrating, querying, and analyzing data across organizational boundaries. Its cost is concrete: analysts spend 40% of their working hours loading and cleaning data, and 60–70% of enterprise data is never used for analytics. The difficulty scales with the schema: enterprise systems routinely expose more than 10^5 attributes under opaque, proprietary vocabularies that no analyst—and no publicly trained model—can resolve by lookup. This dissertation asks how foundation models change the treatment of semantic heterogeneity: which data-management tasks they help solve, and how they compare against four decades of schema-matching and data-integration techniques. It answers through three systems, each closing a distinct gap.

Chorus brings foundation models to data discovery, unifying table-class detection, column-type annotation, and join-column prediction in a single prompt-based architecture with an *anchoring* mitigation against hallucination; it matches or exceeds task-specific deep models trained on 10^5–10^6 labelled examples, reaching 0.93, 0.89, and 0.90 F_1 on the three tasks with no per-task training. Goby confronts the enterprise gap that public benchmarks conceal: it releases the first data-integration benchmark drawn from real, semi-private enterprise data, quantifies the performance drop that both LLM- and ML-based methods suffer on it, and closes that drop by learning a working ontology from data samples and serializing its full hierarchy into the model. *Query Blueprints* attacks the entanglement of structure and vocabulary in natural-language-to-query translation: by letting the user specify query structure exactly and grounding only the unknown vocabulary in data, it answers queries over opaque schemas where end-to-end LLM agents collapse—34% accuracy versus under 10% for four state-of-the-art agents on a scrambled-schema benchmark, at $0.40 versus $13.03 in cost.

Together the three systems trace the contemporary heterogeneity stack: foundation models can replace task-specific training on public schemas, but enterprise data demands that structure be factored from vocabulary, and that vocabulary be grounded in data rather than recalled from training.

Contents

Chapter 1

Introduction

A hospital stores a record for every patient it treats. So does the laboratory that runs the blood tests, the pharmacy that fills the prescriptions, and the insurer that pays for them. All four are describing the same person, the same illness, and the same course of treatment—but each wrote its records for its own purposes, using its own words. One calls the field `DOB`, another `birth_date`, a third `dt_nasc`. One stores the prescribing physician in the same record as the prescription; another keeps physicians in a separate file and links the two by a number. Nothing is wrong with any of these choices. They simply were not coordinated, and no one ever intended them to be combined. When someone finally wants to ask a question that spans all four—which patients on this drug were later readmitted?—the work of reconciling those descriptions falls to a person, and it is slow, manual, and error-prone. This dissertation is about that work, and about whether the current generation of artificial-intelligence systems can do it.

Stated more precisely: data of this kind is kept in a *database*, and the description of how a database is organized—which tables exist, which columns each has, what each column means—is its *schema*. A *query* is a formal expression, written in a language such as SQL, that retrieves an answer from a database; writing one correctly requires knowing the schema, because the query must name the tables and columns it touches. This is the difficulty. Two databases covering the same domain will not share a schema, and the mismatch is not merely cosmetic. The same concept appears under different names, so no amount of string comparison identifies it. The same fact is laid out differently—one row here, a join across three tables there—so even a perfect dictionary of names would not tell a system how to reassemble the fact. And names are frequently not words at all, but internal codes whose meaning lives only in an organization's institutional memory. This condition has a name in the database literature: *semantic heterogeneity*. It is the oldest unsolved problem in data integration, and the community has attacked it for four decades under headings such as *schema matching* (finding which column in one schema corresponds to which column in another) and *column-type annotation* (labelling a column with what its values actually mean—`birthDate` rather than merely "a date").

By the standard academic measures, this problem now looks close to solved. *Foundation models*—large neural networks trained on broad corpora of text, which generalize to tasks they were never explicitly trained for [11]—have driven natural-language-to-query translation above 90% accuracy on benchmarks such as Spider [117], and a single prompted model now annotates columns, detects table types, and matches schemas at the level of systems that were, until recently, trained for each task in isolation. Yet the same models, pointed at a production data warehouse of only 17 tables, answer 22% of queries correctly [37]. The gap between the benchmark and the warehouse is the subject of this dissertation.

1.1 Semantic Heterogeneity

The cost of semantic heterogeneity is concrete and quantifiable. A 2021 Anaconda survey of data-science professionals found that analysts spend 40% of their working hours loading and cleaning data [3]. Despite that effort, Forrester reports that 60–70% of the data inside a typical enterprise is never used for analytics at all, remaining as *dark data* [44, 47, 118]. The reason is the one above: no analyst writes a query against a schema they do not understand, and enterprise schemas long ago stopped being comprehensible to any single person. A typical deployment of Oracle eBusiness Suite or Microsoft Dynamics AX exposes more than 100,000 attributes [13]; SAP ERP installations contain upwards of 90,000 tables under opaque abbreviated names [89]; the MIMIC-II clinical database ships with a 76-page data dictionary purely to document its schema [19].

Three interlocking phenomena define the problem, and they recur throughout this dissertation.

- **Lexical mismatch.** The same concept appears under different identifiers in different schemas (`salary` vs. `wages`; `wdt:P31` vs. `type_of`). Schema designers do not coordinate, and even within one organization, conventions drift over time.
- **Structural mismatch.** The same fact may be modeled as a single table in one source and as a multi-hop join in another. Translation between sources is not, in general, isomorphic at the schema level—a correspondence between two schemas can be a path rather than a pair.
- **Vocabulary opacity.** Enterprise schemas rest on internal jargon, abbreviated codes, and proprietary class hierarchies that appear in no public corpus. A model trained on public data cannot resolve them by lookup, because there is nothing to look them up in.

A recurring instrument against all three is an *ontology*: an explicit, agreed vocabulary of classes and the relationships among them—that a `Physician` is a kind of `Person`, that a `Prescription` has a prescriber. Public ontologies such as DBpedia and Wikidata supply such a vocabulary for general knowledge.

The most ambitious attempt to make that instrument universal was the Semantic Web [8], which proposed that publishers annotate their data against shared, curated vocabularies so that machines could reconcile sources without human mediation. It did not take. The vision survives in pockets—Wikidata is the most visible—but a quarter of a century on, the overwhelming majority of the world's structured data carries no such annotation, and the analyst reconciling four hospital systems is no better off than before. The reasons are worth naming, because they recur throughout this dissertation. Agreeing on a vocabulary is expensive, and the cost falls on the publisher while the benefit accrues to the consumer. A vocabulary rich enough to describe a domain precisely is too large for anyone to adopt correctly; one small enough to adopt is too coarse to be worth adopting. And schemas drift faster than any standards body can ratify, so a mapping authored today is wrong next quarter. Heterogeneity is not a defect that better standards eliminate. It is what data looks like when it is produced independently by parties with their own purposes, which is how nearly all data is produced.

The lesson this dissertation takes from that failure is one of direction. Ontologies are useful, but as artifacts *discovered* from data after the fact rather than mandated in advance—and every technique developed here works bottom-up for that reason. Enterprises rarely have an ontology at all, and when they do it is unwritten; Chapter 4 takes learning one from data samples as its central difficulty, and Chapter 5 takes the same stance toward query vocabulary.

Three questions structure this dissertation: *how do modern generative-AI approaches address semantic heterogeneity, which data-management tasks do they help solve, and how do they compare against traditional methods?*

1.2 The Promise and Its Limit

Foundation models seem, at first, to dissolve heterogeneity by knowing more than any schema could teach. A model that has read the public web already knows that a column of `ISBN`s holds book identifiers and that `dbo:birthPlace` relates a person to a city; it resolves vocabulary from *parametric knowledge*—the facts absorbed into the network's weights during training, available without any lookup at query time—with no supervision and no labelled examples. On the public schemas that fill academic benchmarks, this works, and it works well. It is the reason a single prompt now matches systems that were trained on 10^5–10^6 hand-labelled columns.

The limit is structural, not incidental. Parametric knowledge is bounded by the training data, which is overwhelmingly public [74]; enterprise schemas, by definition, are not. Worse, a model asked about a schema it has never seen does not abstain—it produces a fluent, confident, wrong answer. This behavior is called *hallucination*, and it is not a defect of any particular model: under mild assumptions, a calibrated language model *must* hallucinate at a bounded rate [58], so it cannot be trained away. On an opaque schema the failure is silent. An invented predicate or a hallucinated join key does not raise an error; it returns plausible results that are simply incorrect, and nothing in the output marks them as such. The very fluency that makes these models useful on public data makes them dangerous on private data.

1.3 Thesis Statement

This dissertation argues that the resolution lies not in scaling the model but in changing how it is used:

> *Foundation models advance semantic data management only when they are grounded in data. On public schemas, their generalization replaces a generation of task-specific models. On private, opaque enterprise schemas, they succeed only when vocabulary is resolved from the data rather than drawn from parametric knowledge, and when the structure of a query is factored apart from its vocabulary.*

Two terms in that claim carry the weight. To *ground* a decision in data is to settle it by consulting the database itself—sampling values, probing which paths exist, checking a candidate against instances—rather than by asking the model to recall what an identifier means. To *factor structure from vocabulary* is to separate the two questions that any query implicitly answers at once: what the query should compute, which is a matter of the user's intent, and what this particular schema calls the things it computes over, which is a matter of schema knowledge. The first question the user can answer; the second they usually cannot. Systems that couple them force the user to supply knowledge they lack, and force the model to guess at intent it was never given.

The argument is made constructively, through three systems: Chorus for data discovery on public schemas, Goby for measuring and closing the enterprise gap in column annotation, and *Query Blueprints* for querying schemas whose vocabulary is unknown. Each takes one step away from reliance on parametric knowledge and toward grounding in data, and each is evaluated against the methods it aims to replace.

1.4 Contributions

1. **Chorus establishes what generalization alone can achieve** (Chapter 3). Before an analyst can query a data collection, they must know what is in it. That is the province of

data discovery, and it decomposes into concrete tasks: given a table of electric cars, recognize that its rows are instances of `ElectricVehicle` (table-class detection); label its columns `Manufacturer`, `Model`, `VIN` (column-type annotation); and, given a second table, predict which pair of columns the analyst would join them on (join-column prediction). Each had been solved separately, by a separate deep model trained on 10^5–10^6 labelled examples. The difficulty in replacing them with one prompted model is twofold: the three tasks are mutually informative but a prompt has no mechanism for passing information between them, and a model free to emit any string will emit class names that do not exist in the target ontology. CHORUS answers the first with a unified architecture in which each task's output enters the shared context of the next—knowing the table is about electric vehicles is what disambiguates a column named `Model`—and the second with *anchoring*, a mitigation that checks each output against the supplied ontology and repairs it when it falls outside. With no per-task training, CHORUS matches or exceeds the task-specific models, reaching 0.93, 0.89, and 0.90 F_1 on the three tasks. On public schemas, a foundation model's general knowledge substitutes for task-specific supervision rather than merely supplementing it. CHORUS was nominated for the VLDB 2024 Best Research Paper Award, finishing as the runner-up.

2. **GOBY shows that this success is partly an artifact of the benchmark, and closes the gap** (Chapter 4). Every result in the preceding paragraph, and nearly every result in the LLM-for-data-management literature, is measured on public benchmarks—which over-represent exactly the government, geography, and web-table domains that foundation models have already memorized. Testing that suspicion requires enterprise data that is simultaneously real, expertly labelled, and legally releasable, and those three requirements are what has kept such a benchmark from existing: genuinely private data cannot be published, and anything publishable has usually been public long enough to be trained on. GOBY threads that needle with 1,187 tables scraped from a production event-management workload under a domain-expert ontology, and is, to our knowledge, the first data-integration benchmark drawn from real, semi-private enterprise data. On it, both LLM- and ML-based methods regress sharply. The regression is recoverable, but not by any method that assumes the ontology is given, because in the enterprise it never is: the fix is to *learn* a working ontology from data samples and serialize its full hierarchy into the model. Notably, presenting the entire hierarchy outperforms retrieving only the relevant subset—the opposite of standard retrieval intuition. The contribution is an honest benchmark together with the mitigations the benchmark demands.

3. ***Query Blueprints* answers queries when memorization is unavailable by construction** (Chapter 5). A *query blueprint* lets the user specify a query's structure exactly, in ordinary query syntax, while leaving unknown vocabulary as natural language: *find the* `'companies'` *whose* `'industry'` *is* `'advertising'`, with the structure formal and the three quoted fragments unresolved. The system's remaining job is a pure semantic-mapping problem, and it is harder than a dictionary lookup in two ways. The translation is not isomorphic—a single natural-language edge may resolve to a multi-hop path through the schema, so the search space is over paths rather than over identifiers—and there is no training data for the target schema and never will be. *Blueprint mining* solves it bottom-up: the system generates its own grounding examples, probes the database around each one, mines candidate patterns by random-walk scoring, and selects a consensus translation by instant-runoff voting, confining the language model to a single narrow role. On schemas whose identifiers have been scrambled to neutralize parametric knowledge, this factoring beats four state-of-the-art end-to-end agents by an order of magnitude—34% accuracy against under 10%—at roughly one-thirtieth of the cost.

1.5 The Three Systems

1.5.1 CHORUS: Foundation Models for Data Discovery

CHORUS [62] applies foundation models to three core data-discovery tasks: *table-class detection* (identify the ontology class a table represents), *column-type annotation* (label each column with a semantic type), and *join-column prediction* (suggest the join keys between two tables). Prior work addressed each task individually, with task-specific models trained on 10^5–10^6 labelled examples [54, 98, 116]; CHORUS replaces the stack with a single foundation model and zero- or few-shot prompts.

The system has three design properties relevant to the heterogeneity question. First, it serializes table data and metadata as natural-language prompts—no per-task training is required, and adding a new task is a prompt-engineering exercise. Second, it uses a *unified architecture*: each task's prompt sits in a shared context, so the output of table-class detection (e.g., `ElectricVehicle`) flows downstream as context for column annotation, where it disambiguates columns like `Make` and `Model`. Third, it deploys the *anchoring* mitigation: outputs are checked against the supplied ontology and repaired when they fall outside it, neutralizing a common class of hallucinations.

Empirically, CHORUS matches or exceeds the state-of-the-art representation-learning models on the three tasks while using a fraction of the engineering effort. For public-data benchmarks, foundation-model generalization can substitute for task-specific training. It does not, however, address the enterprise setting.

1.5.2 Goby: Quantifying and Closing the Enterprise Gap

GOBY [63] confronts the enterprise gap directly, and its contribution is twofold.

The first part is a benchmark. GOBY releases a 1,187-table semantic-column-annotation benchmark derived from a production workload in the event-management industry, together with a domain-expert ontology. Datasets sit on a spectrum from fully public (VizNet [50], GitTables [52], T2D) to fully private. Public benchmarks over-represent government and geography tables; private datasets are inaccessible to academic evaluation. GOBY occupies the middle of the spectrum: it is scraped from real APIs with private labels, but is releasable.

The second part is a set of mitigations. On GOBY, LLM-based and ML-based methods both lose approximately 0.10 F_1 points relative to their performance on (semi-)public benchmarks. The mitigations close the gap by treating the ontology as something to be *learned*, not given: an iterative procedure synthesizes a working ontology from data samples, and the synthesized ontology is encoded into the model via *tree serialization*, in which the full class hierarchy is presented as a flattened set of root-to-leaf paths. Across the variants tested, presenting the full ontology context outperformed selectively presenting only the relevant subset, contrary to standard retrieval intuitions.

1.5.3 Query Blueprints: Factoring Structure from Vocabulary

The starting observation of *Query Blueprints* is that a natural-language query implicitly answers two questions at once: *what* the user wants to compute, and *what the schema calls those things*. Coupling them is the source of the worst failure modes on opaque schemas.

A *query blueprint* is a query in the host language—SPARQL, in our setting—where some predicates and constants are replaced by natural-language descriptions. The user specifies the structure (joins, filters, groupings) in formal syntax; natural language is used only where the user does not know the schema's vocabulary. The system's task reduces to binding each natural-language string to a schema-correct predicate or entity, possibly expanding a single edge into a multi-hop path.

The mapping is performed *bottom-up*, by what we call *blueprint mining*. The system generates a small number of grounding examples for the natural-language variables, probes the knowledge graph to extract virtual subgraphs around each example, mines candidate query patterns from those subgraphs via random-walk scoring, and selects a consensus translation via instant-runoff voting. LLM use is confined to seed-example generation; the remainder is classical database machinery—schema matching [91] and keyword search [2, 49]—reorganized for the LLM era.

The design choice has a measurable consequence. On WikiScramble, a benchmark of 218 blueprints derived from real SPARQL queries against a scrambled-identifier version of Wikidata, Query Blueprints achieve 34% accuracy at recall threshold $\tau = 80\%$; four state-of-the-art LLM agents score below 10%. The cost gap is comparable: \$0.40 versus \$13.03 for the entire benchmark. On Goby, Query Blueprints achieve 28% correctness versus 8% for a Gemini-based SQL agent. When the LLM cannot rely on memorized schema vocabulary, factoring structure from vocabulary—and grounding the latter in actual data—outperforms an end-to-end LLM by a wide margin.

The arc. A through-line connects the three, each grounding more of the problem in data than the last. Chorus shows how far parametric knowledge alone reaches on public schemas. Goby confronts the schemas where it runs short, and extends the model's reach by learning an ontology from the data itself. *Query Blueprints* completes the arc, grounding vocabulary in data by construction so that translation holds even where parametric knowledge offers nothing. Each inherits the question the previous one leaves open.

Chapter 2

Background and Related Work

2.1 The Semantic Heterogeneity Problem

Semantic heterogeneity is the long-standing problem that the same real-world concept is represented in different ways across databases, and that the same syntactic element may carry different meaning in different schemas [28]. It is the principal obstacle to integrating, querying, and analyzing data across organizational boundaries, and it persists at every scale at which structured data is stored.

The cost of the problem is concrete and quantifiable. A 2021 survey by Anaconda of data-science professionals found that analysts spend 40% of their working hours on data loading and cleaning [3]. Despite this effort, Forrester reports that 60–70% of data inside a typical enterprise is never used for analytics, remaining as *dark data* [44, 47, 118]. The principal reason is that no analyst can write a query against a schema they do not understand, and enterprise schemas are no longer comprehensible by any single human. A typical deployment of Oracle eBusiness Suite or Microsoft Dynamics AX has more than 100,000 attributes [13]; SAP ERP installations contain upwards of 90,000 tables with opaque abbreviated names [89]; and the MIMIC-II clinical database ships with a 76-page data dictionary just to document its schema [19].

Three interlocking phenomena define the problem.

- **Lexical mismatch.** The same concept appears under different identifiers in different schemas (`salary` vs. `wages`; `wdt:P31` vs. `type_of`). Schema designers do not coordinate, and even within a single organization, conventions drift over time.
- **Structural mismatch.** The same fact may be modeled in one table in one source and as a multi-hop join in another. Translation between sources is not, in general, isomorphic at the schema level.
- **Vocabulary opacity.** Enterprise schemas typically rest on internal jargon, abbreviated codes, and proprietary class hierarchies that are not present in any public corpus. A model trained on public data cannot resolve them by lookup.

The remainder of this chapter surveys the four decades of work that precede this dissertation, in the order in which the field developed it: the schema-matching and data-integration foundations, the data-exchange formalism, the big-data turn, the two waves of machine learning, and the empirical literature on where foundation models fail.

2.2 Foundations: Schema Matching and Data Integration

The earliest formalization of semantic integration in databases is the schema-matching literature, which seeks correspondences between attributes of two schemas. Rahm and Bernstein's taxonomy [91] remains the canonical reference; it classifies matchers by their evidence (schema-level vs. instance-level, syntactic vs. semantic) and by their composition (individual vs. hybrid vs. composite). Cupid [73] is the prototypical hybrid matcher, combining lexical similarity with structural propagation over the schema tree. A decade later, Bernstein, Madhavan, and Rahm [9] revisited the area and noted that, despite hundreds of papers, no matcher had achieved fully automatic operation on industrial schemas; human-in-the-loop validation remained mandatory. The matching effort also scales poorly with the size of the schema: an enterprise system with 10^5 attributes admits roughly 10^{10} candidate pairs.

Data integration extends schema matching by adding a query semantics over the matched schemas. Lenzerini [70] provided the formal foundation, distinguishing global-as-view (GAV) and local-as-view (LAV) mediation. In GAV, the global schema is defined as views over the sources; in LAV, each source is defined as a view over the global schema. The Doan, Halevy, and Ives textbook [29] consolidates the area and names the curation cost as the central bottleneck. The shared limitation of these foundational works is the precondition: every approach assumes that a mediated schema and the source-to-mediator mappings have been authored in advance. None of them addresses how to discover these from data.

Doan and Halevy's 2005 survey of semantic integration [28] is the canonical snapshot of the field as it stood a decade before LLMs. They organize matching techniques into two families. *Rule-based* matchers (TranScm, DIKE, ARTEMIS, MOMIS, Cupid) encode hand-crafted heuristics over schema-level evidence: element names, types, structural roles, integrity constraints. They are cheap, fast, and require no training, but cannot exploit data instances and are difficult to extend to complex or textual schemas. *Learning-based* matchers (SEMINT, LSD, iMAP, Autoplex, Automatch) train classifiers over both schema and data evidence. LSD, in particular, pioneered learning to match hierarchical XML schemas; iMAP reformulated the discovery of complex matches (e.g., `name = concat(first, last)`) as a search problem in an effectively infinite space of candidates. By 2005 the consolidated architectural direction was the *multi-matcher* system: an ensemble of individual matchers, each exploiting a different kind of evidence, whose predictions are combined either by hand or by a learner.

The survey closes with six open problems that, two decades later, remain the right yardstick for evaluating the LLM-era literature. (1) *User interaction*: practical matchers need human feedback, but deployed systems irritate users by asking too many questions. The open challenge is to minimize the questions while maximizing their impact. (2) *Reasoning with imprecise matches at scale*: P2P and web-scale settings will produce thousands of mappings that no human can verify; what can be done with mappings that are known to be only approximately correct? (3) *Schema integration*: once matches are found, merging the schemas into a global one (Batini, Lenzerini, and Navathe, 1986) remains harder than discovering the matches. (4) *Data translation*: elaborating a match into an executable mapping (in SQL, XQuery, GAV/LAV/GLAV) is itself a research problem; Clio was the first system to attempt it end-to-end. (5) *Mapping maintenance*: sources change, and mappings must evolve with them. The 2005 survey calls this "relatively little attended to"—the same is largely true today. (6) *Industrial-strength matching*: a recurring concern that the academic literature was not solving the right problems for industrial schemas; the work of Bernstein et al. [10] on industrial-strength matching and of Rahm, Do, and Massmann [92] on large XML schemas anticipated what is now the central concern of the field. Every one of these problems is, in a recognizable form, exactly what LLM-based methods will attempt to solve.

2.3 Data Exchange

Data exchange formalizes a strictly narrower problem than data integration: given source data and a set of source-to-target dependencies, materialize a target instance that satisfies the dependencies. Fagin, Kolaitis, Miller, and Popa [35] established the foundational semantics. Their starting observation is that a data-exchange instance can admit *many* valid solutions—the dependencies only partially constrain the target, and many distinct instances satisfy them. Choosing one to materialize requires a principled criterion.

The criterion they identify is the *universal solution*: a target instance that is, in a precise algebraic sense, the "most general" one consistent with the dependencies. Every other valid solution can be obtained from a universal solution by specializing it (formally, by a homomorphism). Universal solutions are exactly the right thing to materialize: they encode the full space of valid solutions and add no extraneous facts. The paper shows that, when the source-to-target and target dependencies satisfy a syntactic condition called *weak acyclicity*, a canonical universal solution can be computed in polynomial time by the classical *chase* procedure. Even though the chase had been used in dependency theory since the 1970s, Fagin et al. were the first to identify it as the right machinery for data exchange and to prove that its output stands in homomorphism to every valid solution.

For query answering, the paper adopts the *certain answers* semantics borrowed from incomplete-database theory: the answer to a query against the target is the intersection of the answers to that query over all valid target instances. The certain answers are independent of which solution was materialized, which is precisely the property a principled semantics needs. For conjunctive queries (and unions thereof), the paper shows that simply evaluating the query against the canonical universal solution returns the certain answers. The framework, in other words, supports query answering against the materialized target alone—a property required in settings where source access at query time is impossible (autonomy-preserving systems, peer-to-peer applications).

Once queries become richer the picture darkens sharply. For unions of conjunctive queries with even a small number of inequalities per disjunct, computing the certain answers becomes coNP-complete. More subtly, the paper proves a *first-order inexpressibility* result: there exist queries whose certain answers can be computed in polynomial time, yet no SQL query whatsoever computed against the canonical universal solution returns them. The semantic "right answer" may exist and be tractable, but no relational engine can produce it on the materialized target. This is the formal counterpart of a problem now familiar in the LLM era: an end-to-end system may generate a syntactically valid query that returns plausible-looking results without computing the intended semantics, and the failure can be invisible to the user.

Data exchange subsumes both LAV and GAV data integration: source-to-target tuple-generating dependencies generalize both, yielding the so-called GLAV setting. The principal difference is that data exchange must materialize a finite target, while data integration only answers queries on demand against the sources. Both frameworks, however, share the same precondition: the dependencies are taken as input. Discovering them—whether from schema, data, examples, or natural language—is outside the formalism. The Clio project [29] was the first attempt to derive source-to-target dependencies semi-automatically from correspondences, and the data-exchange paper was explicitly motivated by formalizing the intuitions Clio operationalized.

2.4 Big Data Integration and Dataset Discovery

The mid-2010s saw a qualitative shift in the integration problem. Where traditional data integration concerned itself with a handful of curated sources, *big data integration* confronts millions of

autonomous, dynamic sources whose schemas and content change constantly. Dong and Srivastava's monograph [31] is the canonical statement of this shift. It frames the problem along four "V" dimensions familiar from the broader big-data literature, each of which breaks an assumption that traditional integration takes for granted.

Volume refers not only to the size of each source but to the number of sources, which can run into the millions even within a single domain. *Velocity* describes the rate at which sources update—and the rate at which sources themselves appear and disappear, so that the set of sources to be integrated is itself non-stationary. *Variety* captures the extreme structural and semantic heterogeneity between sources covering substantially the same entities. *Veracity* captures the wide variation in source quality, coverage, accuracy, and timeliness. A central empirical insight of the book is that sources are not independent: many web sources *copy* from each other, often partially and often imperfectly, which means naive voting across sources systematically overweights the loudest values rather than the most reliable ones.

The book organizes the field as a three-step pipeline. *Schema alignment* produces a mediated schema, attribute matchings to it, and schema mappings used to reformulate queries; this is the data-integration component, now stressed by the variety and velocity dimensions. *Record linkage* partitions records so that each partition refers to a single real-world entity; the volume challenge here is the quadratic cost of pairwise comparison, attacked with blocking, MapReduce-parallelized BlockSplit, and meta-blocking. *Data fusion* resolves disagreements among sources to a single truth, using source-accuracy estimates, Bayesian models of value correctness, and the copying-detection machinery already mentioned. Each stage gets a chapter in the book, and each chapter is structured the same way: a quick tour of the traditional technique, followed by what changes when the four V's are imposed.

The book's empirical case studies (the `Flights` domain runs as a thread through the entire book) make the magnitude of the problem visible: tens to hundreds of thousands of sources per domain; K-coverage analyses that show no single source covers more than a fraction of any real entity set; and quantitative characterizations of how few of the trillions of HTML tables on the web are "high-quality" enough to integrate. The book's emerging-topics chapter foreshadows what would become the LLM-era research agenda: *source selection* (choose which of millions of candidate sources to ingest), *source profiling* (characterize a source's content before integrating it), and *crowdsourcing the workflow*. The book pre-dates the LLM era, but its diagnosis—that schema alignment is the limiting step at scale, and that traditional methods do not generalize there—is precisely the gap that LLM-based methods now target.

A line of systems was built on this premise even before the LLM wave. WebTables [15] extracted tables from the web at corpus scale and introduced the three downstream tasks of schema auto-completion, attribute synonym finding, and join-graph traversal. Earlier, wrapper induction [67] addressed the related problem of extracting structured records from heterogeneous web pages. The lineage continues today in data-lake discovery: unionability search [65], joinability search [125], and indexing for correlated-dataset search [94]; end-to-end systems integrate these primitives into ingestion and profiling pipelines [16]. Recent surveys lay out the open problems in dataset search [17, 36, 79]: more natural query languages, better data integration, and incorporating external knowledge.

Industrial systems in this space include Databricks Lakehouse [14], Sigma Computing's WarpGate [22], and Google Dataset Search [12]. Despite their adoption, none provides a complete solution to the heterogeneity problem; they retrieve candidate datasets but defer the semantic mapping back to the user.

2.5 Pre-LLM Machine Learning for Tabular Data

The first wave of machine-learning attacks on data-integration tasks trained *task-specific deep models* on large labelled corpora of tables. Sherlock [54] predicts semantic types from column values using a multi-input deep network and 686,000 labelled columns drawn from VizNet [50]. Sato [119] extends Sherlock with table-context features. Both produce classifiers whose label set is fixed at training time.

A subsequent generation moved to *representation learning*: rather than training a classifier per task, pretrain a single embedding model and fine-tune for each downstream task. TURL [25], TaBERT [116], DoDuo [98], and TaBBIE [56] are the central works. The reduction in per-task cost is real but partial: each downstream task still requires labelled data and a fine-tuning loop, and the vocabulary remains bounded by the pretraining corpus, which is itself drawn from the same public web tables.

The structural limitation of both generations is the same: the model encodes a closed-world vocabulary. A new ontology class, a new enterprise label, or a private schema requires retraining. Chorus sits one step beyond this line of work, replacing the closed-vocabulary classifier with a foundation model whose vocabulary is unbounded.

2.6 Foundation Models for Data Management

The second wave is built on *foundation models* [11]: large language models trained on general text that generalize to tasks they were not explicitly trained on. The mode of operation is fundamentally different from that of the previous section. Tasks are described in natural language rather than encoded as architectures; the model's input and output are text; new tasks require no labelled data, only a prompt.

The argument for foundation models as a substrate for data management was first made as a position. Trummer [108] proposed foundation models for data profiling, building on earlier evidence that LLMs can predict column correlations from names alone [107]. Narayan et al. [77] extended the argument to the data-wrangling tasks of entity matching, error detection, and data imputation. Evaporate [6] revisited wrapper induction with LLMs, demonstrating that a foundation model could discover extraction templates that previously required hand-crafted wrappers.

Each of these works addressed a single task with a single LLM and an ad-hoc prompting strategy. Chorus [62] is the first system to integrate the three core data-discovery tasks—table-class detection, column-type annotation, and join-column prediction—under one architecture with information flow between tasks and explicit risk mitigations (anchoring against an ontology). Where Narayan and Evaporate treat the LLM as an oracle for one task, Chorus treats it as a substrate for an ensemble of tasks whose outputs cross-condition each other.

The advantages of this family over schema matching and data integration as traditionally practiced are structural. First, it requires no per-task labelling: zero- or few-shot prompts replace tens of thousands of training examples. Second, it handles long-tail tasks naturally: any task expressible in natural language is admissible. Third, it unifies across tasks: a single model performs column annotation, join prediction, and query translation under one architecture. The disadvantages are equally structural, and are the subject of the next section.

2.7 The Enterprise Gap and Benchmark Contamination

The empirical literature on LLMs for data tasks rests almost entirely on public benchmarks. VizNet [50] draws 25% of its tables from open government datasets and another 25% from public web tables; T2D is 100% wiki-style tables; GitTables [52] is scraped from GitHub. These distributions over-represent geography, government, and demographic concepts and under-represent the proprietary jargon characteristic of enterprise databases. Prior work has documented the poor generalization of deep-learning techniques across this distribution shift [62].

Two further effects threaten the validity of benchmark-driven evaluation. The first is hallucination. Kalai and Vempala [58] prove that, under mild calibration assumptions, language models *must* hallucinate at a nonzero rate; this is a feature of the generative architecture, not a defect of any particular model. Holtzman et al. [48] document the sampling-time origins of the same phenomenon, and Mallen et al. [74] characterize how factual recall degrades on tail entities. The second is benchmark contamination: Dodge et al. [30] document widespread leakage of benchmark data into training corpora, and SynthWorlds [43] isolates the resulting parametric-knowledge advantage by comparing model performance on the same tasks instantiated over real versus synthetic entities. Together they imply that strong benchmark numbers cannot be taken at face value, particularly on public data.

Goby [63] addresses both effects. Its semi-private provenance places it outside the training corpora of contemporary LLMs (cf. the historical use of this data only in prior data-curation work [97]); its private labels defeat memorization-based shortcuts; and its mitigations (iterative ontology synthesis, tree serialization) close the gap between public and enterprise performance.

2.8 Natural-Language-to-Query Translation

NL-to-query translation has been the most active application of LLMs to database tasks. On the relational side, Spider [117] is the dominant benchmark; on the graph side, recent benchmarks target SPARQL [84]. State-of-the-art LLMs achieve above 90% on Spider, a number widely cited as evidence that the problem is essentially solved.

The most direct refutation of that narrative comes from Floratou et al. [37]—a paper notable not only for its findings but for its authorship: two competing teams shipping production NL2SQL (Microsoft and the Waii startup) co-wrote it as a "public service announcement" that, in their view, the field's discourse had become disconnected from production reality. The paper is structured as four challenges: schema complexity, ambiguity, benchmarking, and responsible AI.

The schema-complexity numbers are stark. One internal Microsoft financial warehouse has 632 tables, 200 views, and more than 7,400 columns; a commercial product-usage warehouse has 2,281 tables, 65,000 columns, 1,600 views, and almost 50,000 additional columns. Spider and even KaggleDBQA are at least an order of magnitude smaller. On KaggleDBQA's relatively small 17-table, 246-column schema, GPT-3.5 achieves only **22.7%** execution-match accuracy when given the full schema in its prompt. The authors then ablate schema-selection strategies: RAG-based selection achieves 18.9% (worse than full schema), LLM-based selection achieves 21.1%, an oracle that selects the optimal tables achieves 24.3%, and an oracle that selects the optimal tables *and* columns reaches 29.7%. The ceiling, even with perfect schema selection, is 30%. The 22% number has become the most-cited shorthand for the gap between academic and enterprise NL2SQL.

A second finding is that the benchmarks themselves are broken. On KaggleDBQA, two human annotators independently classified **41.1%** of the 272 questions as ambiguous—admitting multiple non-equivalent SQL queries as valid answers. The benchmark, however, ships a single ground-truth

SQL per question. Three categories of ambiguity emerged: ambiguity in the mapping from NL to schema elements (36.6% of ambiguous queries), ambiguity in the mapping to data values (19.6%), and intrinsic linguistic ambiguity (34.8%). The paper also investigates LLMs as ambiguity detectors: GPT-3.5 agrees with humans only 44% of the time, GPT-4 reaches 65%—about the same rate at which humans agree with each other (62%).

A third finding is the *semantic-mismatch* problem: the user assumes the database can answer the question, but it cannot, and a good system should explicitly say so. The authors ran Spider questions against an unrelated KaggleDBQA database (a crime DB). GPT-3.5 correctly returned "cannot be answered" in only 60% of cases—for the other 40%, it hallucinated a SQL query that mixed real KaggleDBQA columns with column names absent from the schema entirely. GPT-4, by contrast, correctly refused **every** Spider question against the wrong database. The gap between models on this task is therefore very large, and underscores that semantic-mismatch detection is itself a model-capability frontier.

The benchmarking section argues that both exact-match and execution-match metrics are inadequate for ambiguous workloads and proposes an *intent-based execution match* that allows result-set supersetting, alternate candidate keys, and ordering tolerance. On Spider with GPT-4, the choice of metric moves accuracy from below 40% (exact match) to above 80% (execution match) to above 90% (intent-based)—making the choice of metric, not the choice of model, the dominant factor in headline numbers. They release Archerfish, an open-source NL2SQL benchmarking framework, to operationalize the new metric.

Finally, the responsible-AI section identifies four production-grade concerns: handling harmful content in databases that legitimately contain it (police, medical), preventing LLMs from generating biased SQL when DBs contain demographics, supporting non-English questions (accuracy drops 3.5–15% across Chinese, Hindi, German, French, Greek), and preventing unintended side effects when an NL2SQL system generates a destructive CRUD statement instead of a query. None of these is addressed by any of the dominant benchmarks.

The retrieval-augmented variants of NL2SQL—Graph RAG [34] and hybrid querying [122]—add an LLM call at query time to enrich the result with retrieved context. They introduce a new failure mode: the retrieved context can itself be hallucinated or misranked, and the LLM is free to override the retrieved evidence with parametric knowledge. *Query Blueprints* occupies the opposite design point: rather than letting an LLM generate the query and then retrieving evidence, the user specifies the query's structure, and retrieval (in the form of example-driven pattern mining) grounds only the vocabulary. The factoring follows the bottom-up tradition rather than the top-down semantic-web vision [8], in which schemas were to be aligned by curating a global ontology.

2.9 Knowledge-Graph Querying and Example-Driven Methods

Knowledge graphs have their own literature on heterogeneity, surveyed by Khan [64]. Question-answering platforms over knowledge graphs [82] map abstract schemas to specific ones, but typically rely on direct predicate and entity lookups rather than on non-isomorphic pattern translation.

A separate family of work mines query patterns from examples. *Query-by-example* systems reverse-engineer queries from user-provided example tuples [57, 76]. *Keyword search over databases* formalizes the task of finding Steiner trees connecting keyword matches in the schema [2, 49, 86, 87]. Both lines accept that the schema vocabulary is unknown to the user and bridge the gap with data.

Query Blueprints sits in this lineage but inverts one fundamental dependency: rather than requiring the user to supply the examples, the system generates them. This is the role to which LLM use is confined—producing seed bindings for the natural-language variables in the blueprint. The

remainder of the algorithm (virtual subgraph extraction, random-walk pattern scoring, instant-runoff voting) is classical database machinery. The system thus represents a deliberate inversion of the LLM-centric trend: the LLM is one component of a larger algorithm, not the algorithm itself.

2.10 Join, Wrangle, and Discovery Tasks

Beyond the canonical column-annotation and NL2SQL tasks, a number of allied discovery and wrangling tasks have been addressed in the literature. Join-column prediction was studied by Yan and He [115] via a learning-to-rank approach over a workload log. CHORUS addresses the same task with a foundation model and a Pandas-completion prompt, eliminating the need for a workload log. Entity-resolution and data-wrangling tasks remain active subjects [77, 98], and the foundation-model approach has produced state-of-the-art results on several of them.

2.11 The Remaining Gap

The state of the art, taken together, is paradoxical. LLMs excel at translating natural language to query languages when the schema is familiar, the vocabulary is in the training corpus, and the task is benchmarked publicly. They fail on enterprise schemas, where the vocabulary is opaque, the schema is private, and the query intent is precise enough that hallucinated joins or invented predicate identifiers produce silent errors. Three specific gaps follow, and the three contributions of this dissertation address one each.

1. **The discovery gap.** Foundation models can perform tasks like column-type annotation and join prediction [98, 115], but no prior system combined the full set of data-discovery tasks under a single architecture with explicit risk mitigations. CHORUS [62] extends the foundation-models-for-data-management lineage (§2.6) by unifying three discovery tasks under one architecture, with explicit mitigations for the LLM-specific risks documented in §2.7.

2. **The enterprise gap.** Public benchmarks [50, 52] systematically over-represent web tables and public-domain ontologies, and therefore overstate the performance of LLM-based methods. GOBY [63] confronts this gap (§2.7) with both a benchmark and a set of mitigations, addressing the limitation that the entire pre-LLM and LLM literature is calibrated on public data.

3. **The structure/vocabulary gap.** NL2SQL conflates two problems: deciding what the query *should compute* (joins, filters, groupings) and resolving *what the schema calls the relevant concepts*. The first is a question of user intent; the second is a question of schema knowledge. Coupling them produces structural ambiguity and, on opaque schemas, catastrophic failure modes. *Query Blueprints* extends the example-driven and keyword-search tradition (§2.9) into the LLM era, factoring the two and using formal syntax for the former and bottom-up grounding for the latter.

Chapter 3

CHORUS: Foundation Models for Data Discovery

Data discovery and exploration are major components of the workflow of analysts and data scientists, yet 60–70% of the data within an enterprise still goes unused for analytics [44], remaining as *dark data* [47, 118]. Recent large language models—*foundation models* [11]—generalize to diverse domain-specific tasks they were not explicitly trained on [5, 51, 109, 111]. We apply them to three representative data-discovery tasks: ① *table-class detection*, ② *column-type annotation*, and ③ *join-column prediction*. An outline of the approach is shown in Figure 3.1. We call the approach Chorus; code for the system and experiments is available at `https://github.com/mkyl/CHORUS`.

Despite their promise, foundation models carry serious risks: spurious generation ("hallucination") [48], factual-recall limitations [74], bias [40], dataset contamination [30], logical shortcuts [95], and fallacies [72]. A central goal of Chorus is to harness foundation models for data discovery while mitigating these risks.

Foundation models permit a substantially different approach from prior work. Rather than task-specific architectures trained on 10^5–10^6 labelled examples [54, 98, 116], the model's inputs and outputs are natural-language text, and performance on domain-specific tasks arises solely by generalization. This yields a **unified architecture**: the three tasks share a single context, so the output of table-class detection (e.g., `ElectricVehicle`) flows downstream to disambiguate column annotations such as `Make` and `Model`.

Contributions We summarize our contributions:

- The first work to use foundation models for the data discovery tasks of table-class detection, column-type annotation, and join-column prediction;
- A novel system, Chorus, whose flexible architecture enables the synthesis of multiple data discovery tasks and the deployment of risk mitigations;
- Task-specific designs that exploit zero- and few-shot strategies and allow information flow between tasks;
- The novel mitigation of *anchoring* to reduce foundation-model risks specific to this domain;
- An empirical validation of Chorus against state-of-the-art baselines across the three tasks.

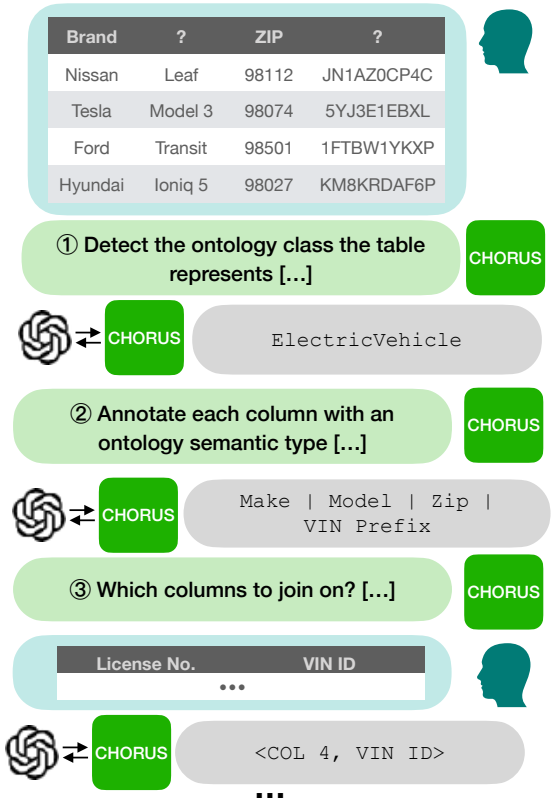


Figure 3.1: Data discovery tasks considered in this chapter. Given an ontology, such as DBPedia, ① we assign an overall type to the table and ② we annotate the columns with semantic types. Last, given another table, ③ we predict the join column. The user provides the data while CHORUS interacts with the foundation model. Data from [80]; full prompts in Figure 3.3.

3.1 Background

3.1.1 Tasks

We assume to be given a *data collection* consisting of a number of relational tables $T_1, T_2, \ldots$. Each table T_i consists of a number of columns, or attributes, $A_1, A_2, \ldots$ and a number of rows, or tuples, $r_1, r_2, \ldots$ The name of a table T_i is, in general, non-informative, for example it may be simply a sequential ID. The columns may optionally have a name $H_1, H_2, \ldots$ or consist only of values.

In addition to the data collection, we are also given a reference ontology of table classes $C_1, C_2, \ldots$, and a reference ontology of column types $\tau_1, \tau_2, \ldots$. For example, the DBPedia.org types for the table classes include `https://dbpedia.org/ontology/Actor` and `https://dbpedia.org/ontology/Continent` and column types include `https://dbpedia.org/ontology/areaTotal` and `https://dbpedia.org/ontology/birthDate`.

We consider three tasks of interest on the data collection:

Definition 3.1 (① Table-class detection)**.** For each table T_i, determine its appropriate class C_j, such that every row $r_1, r_2, \ldots$ represents an instance of the C_j type. We adopt this definition from [66].

For example, table-class detection on the table given in Figure 3.1 could output `ElectricVehicle`, since each row of that table is an instance of that class. Alternatively stated, the table is about `ElectricVehicle`s.

Definition 3.2 (② Column-type annotation)**.** For each table T_i, find a mapping from its attributes (columns) $A_1, A_2, \ldots$ to the reference column types $\tau_1, \tau_2, \ldots$, such that each value in A_i is an instance of the τ_i type. See [1, 25].

For example, column-type annotation on the first column in Figure 3.1 could output `Manufacturer`, since the values are the respective manufacturers of each `ElectricVehicle`.

Definition 3.3 (③ Join-column prediction)**.** Assume an *execution log* L, a history of user actions including table joins and their join conditions, which maps many $(T_i, T_j) \to (A_k, A_l)$ where $A_k \in T_i, A_l \in T_j$. Given two tables T and T', with columns $A_1, \ldots$ and $A'_1, \ldots$ respectively, the *join-column prediction* task is to suggest a pair (A_k, A'_l) of columns such that the equality condition $A_k = A'_l$, which can be used to join the the tables, matches with the choice in the execution log L. For more discussion, see [115].

For example, given the table in Figure 3.1 and another table `car_ registration(name, vehicle_id_number)`, join-column prediction could output `(VIN_prefix, vehicle_id_number)`. The correctness of the prediction depends on the ground truth of which columns the user did in-fact join on.

Ontologies Foundation models contain knowledge of ontologies such as DPBedia.org, Freebase and Wikidata. We focus on universal ontologies, that is, ontologies that aim to represent all entities in general. This is in-line with findings that foundation models encode highly technical knowledge, such as clinical reasoning [96] or electrical engineering principles [102].

3.2 Approach

We outline the structure of Chorus in this section. First, we explore the core idea of ingesting relational data with foundation models and performing data exploration tasks in Subsection 3.2.1. Next, we describe the necessary post-processing and mitigations we develop in Subsection 3.2.2.

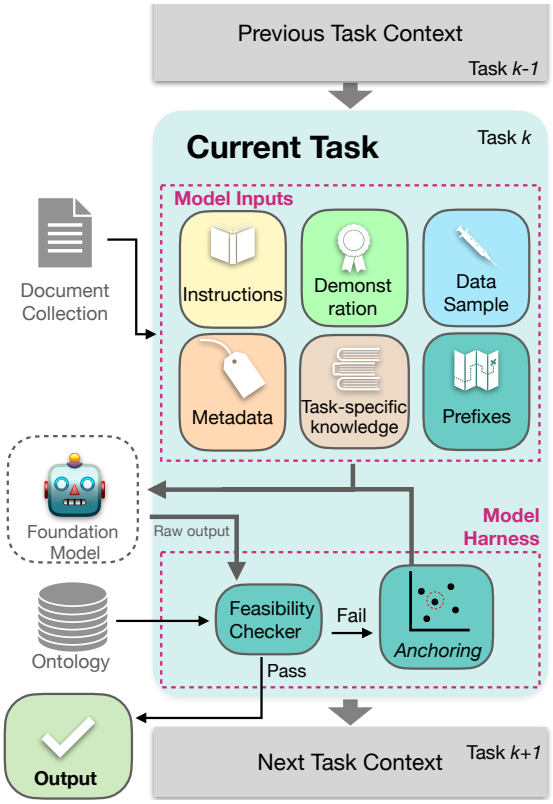


Figure 3.2: CHORUS system architecture.

Figure 3.2 shows the architecture of the system. CHORUS has a unified architecture which runs multiple tasks in the same context, allowing for information flow. Each task is run sequentially, with the output of one task fed as context into future tasks.

For each task instance, CHORUS generates a prompt by concatenating six inputs: context, demonstration, data samples, metadata, task-specific knowledge, and prefixes. They form the "Model Inputs" box in Figure 3.2 and are marked so that they match the corresponding prompt components in Figure 3.3. This natural language input is then fed to the foundation model. The output is then subject to post-processing: checks of parsability and feasibility are conducted. If these pass, the output is extracted. Otherwise, we activate a mitigation process, called *anchoring*, in order to repair the error and prevent its propagation.

3.2.1 Model Inputs

We discuss what inputs are provided to the foundation model and how they are pre-processed and synthesized. We discuss the six components of the Model Inputs module in Figure 3.2, individually. These correspond to the six marked prompt components in Figure 3.3. Once generated, all the above inputs are concatenated to into a single prompt provided to the model.

Instructions A description of the specific task (table-class detection, column-type annotation or join-column prediction) is provided to the foundation model in natural language. These are marked $\mathbf{I}$ in Figure 3.3. For example, we translate the formal Definition 3.1 of the first task, table-class detection, into the English sentence "For the following CSV sample, select one DBpedia.org ontology

Legend: Instruction[I], Demonstration[E], Data sample[S], Metadata[M], Task-specific knowledge[K], Prefix[P]. The superscript letter repeats the colour code, so the figure can be read in greyscale.

For the following CSV sample, select one DBpedia.org ontology that represents the dataset from the following list:[I]
AcademicJournal, AdministrativeRegion, Airline, Airport, [...], University, VideoGame, Work, Wrestler.[K]
For example, for a dataset about hospitals, return `https://dbpedia.org/ontology/Hospital`.[E] Begin your answer with[I] 'https://dbpedia.org/ontology'[P].
```
Brand, ␣, ZIP, ␣,[M]
Nissan, Leaf, 98112, JN1AZ0CP4C,
Tesla, Model 3, 98074, 5YJ3E1EBXL,
[...]
```[S]

(a) Table-class detection

Consider this example.[I] Input:
```
Name, Famous Book, Rk, Year
Fyodor Dostoevsky, Crime and Punishment, 22.5, 1866
Mark Twain, Adventures of Huckleberry Finn, 53, 1884
Albert Camus, The Stranger, -23, 1942
```
Output:
`dbo:author, dbo:title, Unknown, dbo:releaseDate`.[E]
For the following CSV sample, suggest a DBPedia.org Property for each column from the `dbo:`namespace.[I]
``` [...] ```[S]

(b) Column-type annotation

Given two Pandas Dataframes, suggest what `pd.merge` parameters to use to join the dataframes.[I]
df1 =
``` [...] ```
df2 =
``` [...] ```[S]
Complete the correct Pandas merge command.[I] `pd.merge(df1, df2, left_on=[P]

(c) Join-column prediction

Figure 3.3: Prompts used in this chapter, materialized with examples. Most prompt elements are fixed—only the data sample[S] and metadata[M] change for each instance.
```

that represents the dataset." For the third task, join-column prediction, we utilize a code-completion approach. We frame the task as code-completing a Pandas fragment that performs a join, with the code to complete shown in Figure 3.3c. We choose Pandas because it is a very popular framework, with more than millions of example lines of code on the web. This is the *zero-shot prompt* setting: the model can be provided with instructions for a novel task and performs them directly.

Demonstration For the first two tasks, we use the foundation models with task examples as an additional input: this is called the *few-shot prompt* setting. The model is given a few demonstrations of task completion, including inputs and outputs. This is marked **E** in Figure 3.3.

Data sample By serializing the input tables, we can input them into foundation models. For example, consider the example table from Figure 3.1 in the introduction. Serializing the table allows the foundation model to ingest the data. We use the comma-separated values (CSV) format, marked **S** in Figure 3.3.

Because the models have a limited context window size—typically in the few thousands of tokens—tables cannot always be ingested as a whole. Instead, we always serialize a sample of the rows. We find a sample size of five is sufficient. Intuitively, it suffices to consider only a few values to determine column type.

Metadata Schema information including column names (headers) and keys can be incorporated into the input, above the serialized data sample. We found that foundation models can adaptively infer whether the first column of the input is a header or data row, with no modification of the input required. This is marked **M** in Figure 3.3.

Task-specific knowledge For some tasks, additional information can be used to guide the model. For ① table-class detection, if only certain output classes are desired, these can be listed to the model. The model will take these instructions into account when generating an output but they are not hard constraints. The encoding of such additional constraints for the table-class detection task is shown in Figure 3.3a.

Prefixes We also provide the model with *prefixes* with which to complete. This includes the DBPedia format for the table-class detection task and a Pandas code fragment for the join-column prediction task. Both prefixes are marked **P** in Figure 3.3. Prefixes increase the likelihood the model will provide the output in a parsable format rather than deviating into a natural language description.

3.2.2 Model Harness

The foundation model is run within a harness that parses outputs into a symbolic representation and mitigates errors.

Constraint checks Because the model is not constrained in its outputs, it may not always output a feasible answer. In this setting we impose three constraints: table types must belong to the ontology classes, column types must belong to the ontology properties and joins must be on existing columns. An output is infeasible if, in particular, it is not parsable or if it violates any of the three constraints. If this occurs, CHORUS performs anchoring.

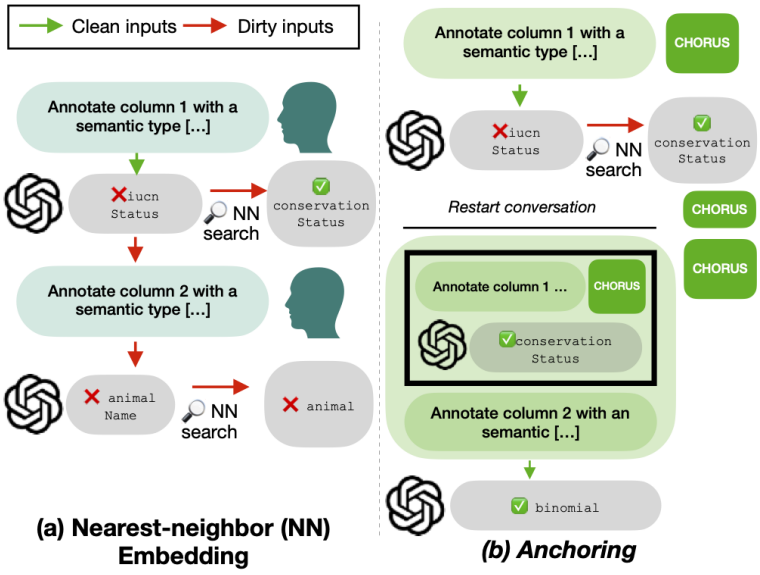


Figure 3.4: *Anchoring* illustrated. The LLM hallucinates an imagined label, `iucnStatus`. Under the standard approach, this poisons all the upcoming tasks; the nearest-neighbor post-processing cannot recover and outputs the incorrect label `animal`. With anchoring, CHORUS intervenes when the first error is detected. A new conversation is started and a *synthesized (false) history* is provided to the LLM, in which it did not make the mistake. With only clean inputs, LLM is able to correctly answer the next task correctly: `binomial`.

Anchoring If the constraints are violated, we do not simply move on to the next task. The risk is of hallucination snowballing [121]: once a foundation model makes a single spurious generation, subsequent outputs are more likely to also be wrong. The model will make mistakes it would otherwise be able to avoid. For example, in Figure 3.4(a): once nonexistent class `iucnStatus` is suggested, another nonexistent class `animalName` follows. Because we maintain context across tasks, we are particularly vulnerable to this.

We call the novel domain-specific mitigation we deploy *anchoring*, shown in Figure 3.4(b). CHORUS ends the conversation when an error is detected. It then initiates a new conversation, feeding the LLM with a false history in which the LLM did not hallucinate. This is possible because the conversational LLM takes as input the full history text, which we can retroactively modify. We insert artificially an existing class from the ontology (e.g. the nearest neighbor in the embedding space to the non-existing class). Fed with this cleaner input, the model is able to directly provide the correct answer.

3.3 Experiments

We empirically evaluate CHORUS on the three tasks defined in Section 3.1.1. For each task, we select a task-specific benchmark and compare with baselines representing the state of the art. Table-class detection ① is evaluated in Section 3.3.1, ② column-type annotation in Section 3.3.2, and ③ join-column prediction in Section 3.3.3. The code for the experiments in available at `https://github.com/mkyl/CHORUS`.

Table 3.1: *Capabilities of related systems.* Only our system supports all studied tasks out-of-the-box and without additional training.

System	Table-class detection	Column-type annotation	Join-column prediction
DoDuo [98]	●	✓	✗
Tabert [116]	●	✓	✗
Sherlock [54]	✗	✓	✗
Trifacta Wrangler [105]	✗	●	✓
Chorus	✓	✓	✓

✓ supported out-of-the-box ✗ no support ● required modification or training-data collection (see text)

Table 3.2: Summary of the datasets used in this thesis. Numbers indicate the size of the data used.

Dataset Title	**# Tables**	**Avg. # Columns**	**Avg. Rows**
T2D-class v2	237	7.41	118
VizNet	∼ 10 600	3.03	5 200
GitNotebooks	24 579	30.9	60 242
Overall	35 416	23.0	43 491

Baselines We considered the following state-of-the-art systems for data exploration: relevant systems include Tabert [116], DoDuo [98], Sato [119], TURL [25], TaBBIE [56], Auto-suggest [115], Trifecta Wrangler [105], Paxata, Tableau Prep, and Sherlock. DoDuo outperforms TURL and Sherlock on column-type annotation [98], so we select it for evaluation. Sato and Sherlock are similar, with Sato utilizing additional signals not found in our benchmarks, so we evaluate the better-established Sherlock. TaBBIE can embed tables but is not trained on column-type annotation unlike DoDuo and Tabert, so we avoid it for the column-type annotation task. Tabert is a work similar to DoDuo and TURL, but from the NLP community rather than the data management community, so we also test it too. For join-column prediction, Trifacta Wrangler outperforms Paxata and Tableau Prep [115]. Auto-Suggest is reported to outperform Trifacta Wrangler, but is a proprietary research project not released publicly. Thus we select Trifacta Wrangler for testing.

For the evaluated prior works Tabert, DoDuo, Trifecta Wrangler and Sherlock [54, 98, 105, 116], we utilize each tool if applicable to the task. If the baseline is not designed for a particular task, but can be straightforwardly adapted, we do so. We describe all modifications in the task subsection and always use established adaptations if available. If the modifications required would be extensive enough to become their own research project, we consider that task unsupported. In all cases, we use the pretrained embeddings without modification, as provided by the authors. Table 3.1 outlines the systems we tested and tasks they support.

DoDuo provides two embedding variants: one trained on the WikiTables dataset and another on VizNet. We label them DoDuo-Wiki and DoDuo-Viz.

Datasets Table 3.2 outlines the three experiment benchmarks we use. For the table-class detection task, we test on the T2D-class v2 dataset [93], a "gold standard" corpus of 237 tables, manually annotated by experts with one of 39 DBPedia.org classes. These tables were in turn selected from the Common Crawl corpus of web tables [38]. For column-type annotation, we sample a subset of the VizNet dataset [50], extracted by the Sherlock team [54], comprised of 32 386 columns with

one of fifteen types from approximately 10 600 tables. This is in line with prior work that uses VizNet [52]. For the join-column prediction task, we use a dataset we call GitNotebooks, extracted by the Auto-suggest team [115]. We select 300 tables from that dataset for which we have join data to run manually. Here we use a sample as one of the baselines, Trifacta Wrangler, does not have an API but instead predictions must be produced manually. Separately, we run all 24 thousand tables on the baselines with an API. For the first two tasks, which require defining a type system for classes and properties, we use the DBPedia ontology [75] for our experiments. This is a community-sourced ontology and is the standard in previous studies.

Setup We use the GPT-3.5 model [83] as it is the most widely-available large model with API access at the time of writing. All other code was run on a commodity laptop with 8 physical ARM cores and 16GB of main memory. Running all experiments came to a total of $20 in API costs.

We evaluate using the metrics *precision*, *recall* and F_1 score. Precision is the proportion of true positive results out of the total predicted positive results, while recall is the proportion of true positive results out of the total actual positive results in the dataset. The F_1 score is the harmonic mean of precision and recall. Since we deal with a multiclass setting, we calculate these metrics for each class separately then aggregate by taking the mean, weighted by the class size. Weighted precision, recall and F_1 are the standard metrics in prior work [15, 54, 98, 119]. We also report average throughput and cost for each task.

3.3.1 Table-class detection

For the first task, ① table-class detection, we tag each table with the DBPedia ontology entry that represents the row-type of the data. Of the 1 000 datasets that comprise the T2Dv2 dataset, 237 tables have table-class correspondences available while 763 do not—we exclude the unlabelled ones from the supervised evaluation. We call this subset of 237 annotated tables *T2D-class v2* and use it for evaluation on this task. We note that only 40 classes are utilized in this "gold standard" mapping, while DBPedia ontology has 769 classes.

We compare against the baselines DoDuo and TaBert. No approach in the prior work provides out-of-the-box capabilities on this task, so we add a classification layer on top of the pretrained embedding layer. After computing the column embeddings using DoDuo or TaBert, predictions are extracted by adding a pooling layer, fed to a multi-layer perceptron, and then finally taking the soft-max. This is a straightforward method of adapting the embeddings to our multi-class setting, used in prior benchmarks for table-class detection [66]. We fix the embeddings to their pretrained values and learn the weights of the classification layer using five-fold cross-validation.

Supervised variant To allow for comparisons with prior work, we initially restrict our system to picking out of the 33 classes. This is because all other approaches require training on labelled instances—the baselines cannot predict outside those classes. We test 33 classes rather than 40 because the classes that occur only once cannot be tested on baselines that require supervised training (DoDuo and TaBERT), since a result requires a disjoint training and test set.

Table 3.3 shows the results. Chorus improves on the three baselines on all metrics. F_1 score is improved by 0.169 points, precision by 17.5 percentage points and recall by 15.5 percentage points. Of the baselines, DoDuo-Wiki provides the best F_1 and precision, while TaBERT provides the comparable recall. The best performing models, TaBERT and DoDuo-Wiki are trained on CommonCrawl, a superset of the T2Dv2 benchmark. DoDuo-Viz which is trained on the VizNet, a dataset disjoint from T2Dv2, has the weakest performance. The numbers for TaBERT are in line

Table 3.3: Weighted F_1 scores for *table-class detection* on T2Dv2 dataset. Systems are compared with the expert-annotated classes for each table. The $n = 237$ tables each correspond to one of 33 `DBPedia.org` classes.

	F_1-score	Precision	Recall
DoDuo-Viz	0.654	66.8%	68.3%
DoDuo-Wiki	0.757	78.6%	76.9%
TaBERT	0.746	76.3%	76.8%
Chorus	**0.926**	**96.1%**	**92.4%**

with prior replications [66], while to the best of our knowledge this is the first benchmarking of DoDuo on this task.

Unsupervised variant Next, we relax the classification domain, allowing the foundation model to choose any of the 768 classes of the DBPedia ontology. We then compare the quality of the classes with that of the human-expert labels. DoDuo and TaBERT are not evaluated in this task setting as they cannot predict outside the classes they have observed in training.

For 93% of tables, our system produces correct results. Of that portion, 83 percentage points are comprised of exact matches, while 10 percentage points are *better-than-correct* results. This means we judge the predicted labels are clearly and unambiguously better than those selected by the benchmark authors. This is a strong claim so we list all such datasets in the technical report [61], with evidence. For the final 6% the answer is incorrect: this can mean the answer is wrong or simply worse than the label provided by the expert. This means that on the relations where Chorus and the expert-label disagree, our system is 1.6× more likely to be correct.

Chorus has a throughput of nearly 31 tables per second on this benchmark and cost an average 2.5¢ per 100 table-class predictions.

3.3.2 Column-type annotation

Next, we compare the ability of our system to assign classes to table columns. VizNet is a collection of tables, extracted by the Sherlock [54] team from the VizNet repository [50] of data visualizations and open datasets. VizNet comprises 31 million columns in total, of which a test set of 142 000 can be used for evaluation—the rest have trained on by DoDuo and Sherlock. Of those, we select a subset of 15 classes which are supported by both DoDuo-Wiki and DoDuo-VizNet (these baselines support a disjoint set of classes), arriving at 32 386 test columns used as a benchmark in this section.

Baselines We compare against TaBERT [116], DoDuo [98] and Sherlock [54] on this task. Since Sherlock and DoDuo are designed for column annotation, we use the out-of-the-box model provided by the original teams. We restrict both to the fifteen target classes by setting the probabilities of non-target classes to zero. For DoDuo-Wiki, which supports a distinct set of classes, we perform a manual mapping to the class names used by DoDuo-VizNet and Sherlock. For TaBERT we train an additional classification layer on top of the pre-trained embeddings that these frameworks provide. We fix the embeddings to their pretrained values and learn the weights of the classification layer using five-fold cross-validation.

Results Table 3.4 contains the results for the VizNet dataset. Our FM-based approach improves performance on the measured metrics of F_1-score, precision and recall. The best performing method

Table 3.4: Weighted F_1 scores for *column-type annotation* on VizNet test set, with $n = 32\,000$ columns. Systems are compared with the "gold standard" classes for each column. Methods which are also pre-trained on VizNet are marked with an asterisk $*$.

	F_1-score	Precision	Recall
DoDuo-VizNet$*$	0.876	89.4%	87.2%
Sherlock$*$	0.954	96.2%	94.6%
TaBERT	0.321	32.6%	32.0%
DoDuo-Wiki	0.440	59.2%	45.4%
Chorus	**0.891**	**91.2%**	**88.8%**

is Sherlock, narrowly beating DoDuo-VizNet, with a 0.954 F_1 score. If we consider methods which are not specifically pretrained on VizNet (note, which is also the test set) Chorus is the best performing on all three metrics. It has comparable F_1 and precision to Sherlock, but 6 percentage points lower recall.

Note in particular DoDuo-Wiki, which does not have access to VizNet at pretraining time, has a large regression in performance compared to DoDuo-Viznet, nearly half F_1 points. This drop is in line with previous results, see Section 3.4. We sanity-check the low scores of TaBERT by replicating previously reported scores [66].

Chorus achieves a competitive throughput of 41 columns per second (col/s), comparable to Sherlock's 50 col/s and exceeding DoDuo's 7.3 col/s and TaBERT's 4.5 col/s. This corresponds to benchmark completion in 13 minutes, as contrasted with over 2 hours 10 minutes for TaBERT. The average cost of GPT-3.5 calls for this task was 1.3¢ cents per 100 columns.

3.3.3 Join-column prediction

Finally, we evaluate our approach's ability to suggest which columns are the correct choice for a join, the join-column prediction task. We use the *GitNotebooks* dataset from [115], a collection of 4 million Python notebooks (and their associated relational tables) including 24 thousand joins collected from Github. One of the baselines, Trifacta Wrangler, requires manual execution and recording of each prediction. For that reason we restrict this benchmark to 300 randomly sampled tables.

Baselines For this task, we compare with three baselines. Jaccard similarity, J, is the first. Two columns are selected such that $\text{argmax}_{c \in C^T, c' \in C^{T'}} J(c, c')$ where $J(X, Y) = |X \cap Y|/|X \cup Y|$. This is a commonly used approach in the literature [18, 23, 78, 115]. Another baseline is Levenshtein distance [71], which selects the pair of column names with the smallest edit distance between them. The final baseline is Trifacta Wrangler [105], a commercial product spun off from the Wrangler research line [59]. When joining two tables in this product, it suggests the keys on which to join them. As no API was available, we obtain all Trifacta predictions by joining manually.

Results Table 3.5 shows the quality of estimates for our approach and the baselines. We measure the quality of the predictions by the same criteria as the previous tasks. By these metrics, our approach improves the quality of predictions and beats the next-best approach by a clear margin: F_1 score is improved by 0.072, precision by 8.4 percentage points and recall by 6.0 percentage points. This performance is maintained when scaling to the full dataset. On this task, our system has an average throughput of 23.5 predictions per second and cost approximately 5¢ cents per 100 predicted joins.

Table 3.5: F_1 scores, precision and recall for the *join-column prediction* task on GitNotebooks dataset.

	F_1-score	Precision	Recall
Manually-run subset, $n = 300$			
Jaccard	0.575	60.7%	54.7%
Levenshtein	0.718	72.3%	71.3%
Trifacta Wrangler	0.823	82.6%	82.0%
Chorus	**0.895**	**91.0%**	**88.0%**
Full dataset, $n = 24\,579$			
Jaccard	0.458	63.3%	35.9%
Levenshtein	0.777	78.7%	76.8%
Trifacta Wrangler		No API	
Chorus	**0.912**	**93.2%**	**89.4%**

Table 3.6: Data contamination experiment. Weighted F_1 scores for *table-class detection* on public benchmarks versus tables the foundation model is guaranteed to have not been trained on.

Dataset	F_1-score	Precision	Recall
Public benchmark (VizNet)	0.865	90.1%	86.7%
Guaranteed-unseen	0.857	90.0%	81.8%

3.3.4 Dataset contamination

Here we perform an experiment to validate whether any of the testing data occurred in the training corpus of the large-language model, an issue called *dataset contamination* or *data leakage*. Because these models are trained on internet data [39] and we use public benchmarks, they may have seen the test data in training.

We test on seven guaranteed-unseen tables (listed in the technical report [61]) and their columns, all uploaded between April–June 2023 to the federal data repository Data.gov. They are guaranteed-unseen because the foundation model training was completed on or before March 2023. Repeating the supervised column-type annotation task as in Section 3.3.2, we measure a 0.857 F_1 score, 90.0% precision and 81.8% recall. This is within 0.01 F_1 points, 0.1% precision and 5% recall of the benchmark results. See Table 3.6. The recall falling more is expected, as this reflects the datasets being more diverse and therefore difficult to classify.

3.3.5 System characteristics

Determinism We examine the impact of nondeterminism in the foundation model on the performance of Chorus. The randomness of the generation is controlled by the *temperature* hyperparameter. To assure that the results of Chorus are reliable, we conduct the following experiment: we run the T2D table-class detection benchmark 25 times, five trials for each value of T between $0, 1/4, ..., 1$. Figure 3.5 shows the result. Chorus's performance is consistent: at the ideal temperature setting the F_1 score sees error bars of 0.01 F_1 points. The best performance is obtained at lowest temperature, 0.0—this is in contrast to NLP tasks like summarization that benefit from higher temperatures. In the prior experiments we use the default temperature, as to get Chorus running with minimal hyperparameter tuning.

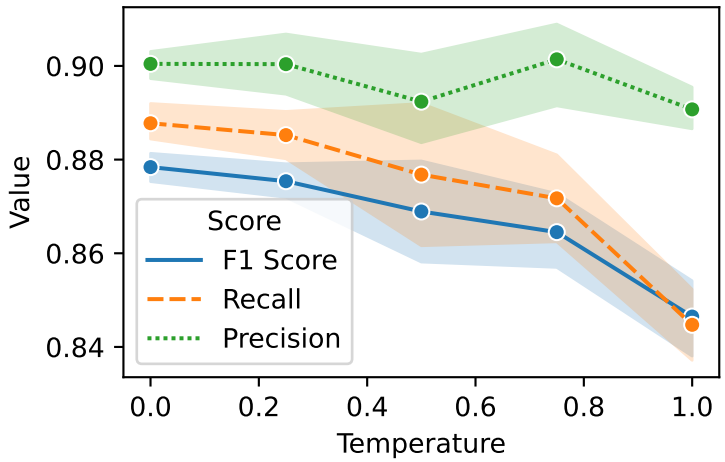


Figure 3.5: *Determinism vs. performance.* We conduct 25 runs of CHORUS on the T2D table class benchmark. Shaded bands indicate confidence intervals. Temperature is a parameter controlling the randomness of the foundation model, with zero being the most (but not completely) deterministic.

Table 3.7: Alternative foundation models. Weighted F_1 scores for *table-class detection* on T2Dv2 dataset, for different choices of foundation model used by CHORUS. Parameter size in brackets. GPT-3.5 numbers identical to experiment in Figure 3.3.

	Table-class correctness		
Model choice	**F_1-score**	**Precision**	**Recall**
GPT-3.5 (175B)	0.926	96.1%	92.4%
LLaMA 2 (70B)	0.893	92.2%	86.5%
Vicuna/LLaMA (13B)	0.713	79.2%	64.1%
Vicuna/LLaMA (7B)	0.713	75.3%	67.5%

Alternative models To demonstrate the versatility of this approach, we run CHORUS with three alternative, open-source foundation models on the table-class detection task. We consider Vicuna [124], a variant of LLaMA [103] at two sizes: 13 billion parameters and 7 billion parameters. The more advanced model is LLaMA 2 [104], the SOTA open-source model with 70 billion parameters.

Table 3.7 shows the results. While OpenAI's GPT model performs best, the best open-source model is very competitive. LLaMA 2 outperforms the best baseline model for this task—DoDuo-Wiki—by 0.136 F_1 points, on precision by 13.6 percentage points and on recall by 9.6 percentage points. This model lags behind the proprietary and larger GPT model by only a modest 0.03 F_1 points. Open-source LLMs are now compelling alternatives on the tested task.

Ablations We conduct ablation experiments to measure the contribution of individual components of CHORUS. We remove one component at a time and note the loss of scores compared to the unaltered model. Figure 3.6 shows the results. First, we remove the demonstration from the prompt. This results in an F1 score loss of 0.03, a recall loss of 4.7 and a precision loss of 4.7 percentage points. Next, we remove the metadata where it is available. This results in a cumulative F1 score loss of 0.04, a recall loss of 5.1 and a precision loss of 5.6 percentage points. After that, we disable anchoring. This results in a cumulative F1 score loss of 0.389, a recall loss of 47.4 and a precision loss

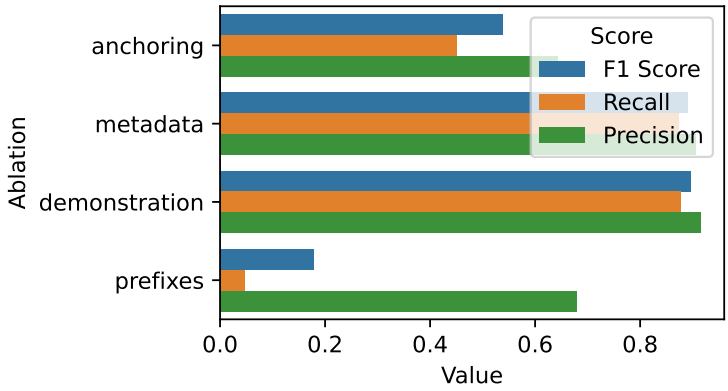


Figure 3.6: *Ablation experiments.* We ablate key features of CHORUS and report performance characteristics on the T2D table-class detection task.

of 31.9 percentage points. From this the prevalence of hallucinations can be gleaned: the substantive score loss implies that hallucination is highly prevalent. The score improvement with anchoring implies that it can be recovered from: the FM mistakes are not severe but rather resemble the ground truth. Finally, we remove the prefixes from the prompt. This results in a cumulative F1 score loss of 0.736, a recall loss of 92.3 and a precision loss of 53.1 percentage points.

3.4 Discussion

Training data collection A major advantage of a foundation-model approach is that there is no need for training on specific tasks. In contrast, TABERT requires 26 million tables for training its embeddings. In [25], the use of 250 labels for one task is considered a "small dataset" by the authors and leads to subpar performance. In contrast, our prompts in Figure 3.3 use zero or none examples for each task.

Out-of-domain performance We note a troubling pattern of a lack of cross-domain generalization in representation-learning approaches. The tested baselines degrade when used to embed tables not from the dataset the embeddings were trained on. This finding is in-line with prior work: regressions of up to 0.40 and 0.30 F_1 points when generalizing to new datasets have been reported [25, 52].

Flexibility Another advantage of CHORUS we observe in the experiments is task adaptability. In the ① table-class detection task, we are able to switch the prediction domain easily. Restricting to the 33 classes used by the benchmark can be done by providing the permitted classes to the foundation model; allowing the model to generalize to other DBPedia classes (the *unsupervised* heading of Section 3.3.1) is as simple as omitting those instructions. Contextual information, such as table title or URL, could be as easily added. Previously, such modifications would require retraining the embeddings.

Limitations and risks We control the risk of dataset contamination by testing for it in Section 3.3.4. The performance of CHORUS on guaranteed-unseen datasets is comparable to those in public benchmarks, so good performance on those benchmarks cannot be explained away as simple data contamination. That test is narrower than it appears, however. Seven tables published after the

training cutoff establish that these particular answers were not memorized; they do not establish that tables of this kind are unfamiliar. Every benchmark used here is drawn from the public web, which is also where the training corpus comes from, so the evaluation cannot separate competence from familiarity. Chapter 4 builds the test that can, on real enterprise data, and measures the regression this one could not detect: 0.18 F_1 points on the same task, using the same underlying method.

A second limitation is that the safeguards presuppose the vocabulary they protect. Both the feasibility checks and the anchoring built on them require a closed, enumerable ontology against which a generation can be declared invalid, and anchoring is what carries the system—removing it costs 0.389 F_1 points (Figure 3.6), more than any component except the output prefixes. Enterprise schemas seldom arrive with such a vocabulary; recovering it is the problem rather than the premise. Chapter 4 has to learn an ontology from data samples before any of the machinery here can be applied. Chapter 5 makes the sharper version of the same point: when schema identifiers are scrambled so that they cannot be recalled from training, agents built on these models answer fewer than 10% of queries. What reads as schema understanding on public data is substantially recall of a public vocabulary.

Two risks do not surface in the numbers at all. Formal linguistic fluency means that errors may fool human reviewers [68, 69], an effect prior work calls *subtle misinformation* [90]; the constraint checks catch a class name that does not exist, but nothing catches a class name that exists and is wrong. And prompts may not be robust to changes [123]—the prompts reported here were arrived at by iterating against these benchmarks, so their tuning is not independent of the scores they produce.

Pattern-matching hypothesis We posit that LLMs outperform representation learning approaches because they are able to extract a large number of linear transformations over the embedding space from their training data and compose them in simple ways. In the full technical report, we conduct an experiment contrasting LLM activations with DODUO weights, supporting this hypothesis [61]. Prior work [81] supports this hypothesis and the limits of LLMS in composing these patterns, generally failing to combine patterns after about four reasoning steps [33].

Future directions *Additional tasks. Additional tasks.* The above hypothesis suggests the promising performance may extend to many more tasks. Related tasks to be explored include *schema auto-completion* [15], where missing parts of a partial schema are suggested to the user; *join-graph traversal*, where successive tables to join on are suggested [32]; and outlier detection, where erroneous data are detected. These are all promising because they involve composing simple patterns that are prevalent in FM's training data. On the other hand, novel approaches will likely be needed to apply FMs to tasks like *data provenance* [42], since these involving tracking more patterns than FM's are thought to currently support. Incorporating proprietary information may also be difficult to due the dearth of these patterns in the training data.

Private or domain-specific datasets As with all the tested baselines, the foundation models are trained on public data. The distribution of data in the public sphere differs significantly from that in specialized domains or private data. It is worth investigating whether the observed capabilities continue to hold on e.g. enterprise data lakes. Further application to domain-specific ontology such as DRON, a pharmaceutical ontology of drugs, would also be interesting to investigate.

Fine-tuned or distilled models Larger models are of interest because if scaling laws continue to hold, their performance should improve [112]. The large inference cost of foundation models has

sparked interest in model distillation: reducing the size of the model without incurring a reduction in specific capabilities [27, 113] but at cost of degraded emergent abilities [45].

3.5 Summary

CHORUS integrates foundation models into data discovery, unifying table-class detection, column-type annotation, and join-column prediction under a single prompt-based architecture. It matches or exceeds the task-specific representation-learning models on all three, and does so without the 10^5–10^6 labelled examples they require. It is also more robust across benchmark datasets than those models, which lose up to 0.40 F_1 points when embedded on tables from a corpus they were not trained on.

Two claims from that work have held up in the years since, and one deserves qualification. The first is that unification pays: information flowing between tasks in a shared context is not a convenience of implementation but a source of accuracy, and it remains the cheapest available gain on these tasks. The second is that the raw model output cannot be trusted. *Anchoring* was introduced here as one mitigation among several, and it has proven the load-bearing one—constraining generation to a supplied ontology removes an entire class of failures that no amount of prompt refinement addresses.

The qualification concerns what the benchmark numbers mean. Section 3.3.4 tests for contamination directly and finds performance on guaranteed-unseen data comparable to the public benchmarks, which rules out the crudest explanation. It does not rule out the subtler one: these benchmarks are drawn overwhelmingly from public web tables, and public web tables are what the model has read. The pattern-matching hypothesis of Section 3.4 predicts as much—if the model is composing patterns absorbed from training data, then a schema whose patterns are absent from that data should defeat it, and no public benchmark can test that. The private-data caveat raised at the end of Section 3.4 was, at the time, a direction for future work. Chapter 4 makes it the object of study, and finds the predicted regression. Chapter 6 takes up what this implies for the field's calibration more broadly.

Chapter 4

Goby: Bridging LLMs to Enterprise Data

Despite intensive academic and industrial interest, the uptake of large language models (LLMs) for data management and integration tasks remains limited in practice. Gartner reports that at least 30% of LLM-based enterprise projects will be abandoned after proof of concept by the end of 2025 [99], with the most common reason being the low quality of results. Yet on public benchmarks such as VizNet [50], T2Dv2 [93], and FDA-510k, LLM-based approaches are reported to be state of the art [62]. This raises the question that organizes the chapter: *what explains the LLM performance gap between research and practice?*

These challenges are not new. An industry survey by Rexer Analytics in 2023 reports that only 32% of machine-learning implementations in the enterprise reach deployment, and among those introducing fundamental architectural changes (such as LLMs) only 22% succeed [4]. A significant reason is population drift: the data on which a model is deployed differ from the data it was trained on. This chapter is a call to action—evaluating LLMs on public data-management benchmarks risks an overly rosy picture of their abilities.

Contributions We summarize our contributions:

- Goby, the first data-integration benchmark drawn from real enterprise data: 4.04 million rows across 23 203 columns of a production event-marketing workload, together with the semantic-type hierarchy its domain experts wrote and the column annotations they applied, released publicly;
- A measurement of the enterprise gap on semantic column-type annotation, showing that both LLM- and ML-based methods lose roughly 0.10 F_1 points on Goby relative to their reported scores on public benchmarks;
- A method for recovering that loss without labelled training data, by learning a working ontology from data samples rather than assuming the model already knows the target vocabulary;
- A comparison of three ways to encode a learned ontology into a language model—grammar-constrained decoding, step-by-step prompting, and full tree serialization—finding that serializing the entire hierarchy into the context is the most robust, at 0.85 F_1 against a baseline of 0.71;
- An analysis of what drives the gap, weighing dataset contamination against distribution shift and assessing the evidence for each.

4.1 Background

Benchmarks and their accompanying datasets can be categorized, in general terms, into two buckets: public and private. On the fully-public end, there are standard benchmarks such as VizNet [50] and GitTables [53]. These contain verbatim public data from web sources. Another well-known example is the Spider Benchmark [117], which compiles databases from sources such as Wikipedia but generates questions over them using the traditional graduate-student-based approach. On the other end, there are private datasets made from proprietary data sources and labeled with proprietary labels, which are not possible to understand outside the originating organization. Our dataset is oriented towards these private datasets.

Further, separate from the issue of public accessibility is the over-representation of some high-visibility domains in benchmarks. Examples of such domains are celebrities, government, and geography. While these domains are very well-represented on the public web, they do not make up the bulk of enterprise data.

Task suitability is another concern. Many existing benchmarks are constructed in a manner not authentic to enterprise use. These benchmarks are created by crawling the web for specific patterns that are easy to identify. For example, GitTables is constructed by searching GitHub for patterns such as `id` and `state`, and similarly, for T2Dv2, tables that were easy to annotate were selected from Wikipedia. In other words, the "how" of data collection dictates the "what" of the database content. On the other hand, organizations generally decide on "what" datasets they have a business need for and then the "how" of data collection.

Virtually all academic systems are evaluated on public datasets (*e.g.*, [7, 62, 77]). Gartner reports that lack of ability to perform evaluation is common in enterprise setting: industry generally does not have its own benchmarks [99]. In personal communication with practitioners building data management systems, we have found this to be true.

Many research systems focused specifically on public data, such as the seminal WebTables system [15]. Prior work noted the poor generalization of many deep learning techniques [62]. The reason for that is the high risk of overfitting to the training data, which restricts the model's ability to generalize and adapt to new, unseen data and contexts. For example, consider the performance of the deep learning model DoDuo [98]. In the setting where it is trained and tested on the same data distribution (VizNet), the performance is strong. However, when we attempt to generalize by training on the similarly structured Wiki dataset and testing it on VizNet, its performance drops by at least 30% in terms of F1-score, precision, and recall [62]. This comparison is merely between two public datasets with very similar data types and structures. An unseen private dataset, which often has unique structures and proprietary labels, may pose an even bigger challenge. This underlines the necessity for more robust approaches to bridge the gap between private and public data sources.

We focus this chapter on a specific data integration task: semantic column type annotation [54, 98, 120]. Given a relational table, column annotation involves assigning to each column a type that corresponds to the type of values stored in that column. The classes are drawn from a set of labels: if those labels are basic types (e.g., date, time), the problem is called *atomic* type annotation, while if the labels have semantic meaning (e.g., birthday, deadline date, best-by date), it is *semantic*. We focus on the latter formulation in this chapter.

4.2 Goby Benchmark

Goby is a benchmark dataset we derive from a production industry workload in the event promotion and marketing setting. The dataset was compiled around 2017 and encompasses over 4 million

Table 4.1: Key characteristics of the GOBY dataset.

GOBY	
# Tables	1 187
Avg. Rows / Table	3 405
Avg. Col / Table	20
Total Rows	4 042 519
Total Columns	23 203

events.

Constructing GOBY The dataset was constructed over several years. First, over one thousand wrappers were written by professional developers in order to convert web pages and APIs into relational tables. Once those wrappers were developed, they were executed to obtain 4.1 million rows of data, each corresponding to an event. Next, the team developed a set of semantic types common to all the events. The labels were applied to 40% of the columns. The rest of the columns were miscellaneous data not of interest to the task. Finally, the semantic type mappings were used to create a universal schema which unifies all the source tables into one.

Benchmark components Its components are:

1. a semantic type hierarchy developed by domain experts
2. nearly 1 200 source tables, each corresponding to the output of one wrapper
3. mapping each annotated with the semantic types of its columns
4. a universal table in which all data from the source tables is unified

Benchmark Characteristics All in all, GOBY is comprised of a total of 4.04 million rows and 23 203 columns. Key characteristics quantifying the dataset are provided in Table 4.1.

GOBY exhibits high task suitability for data integration tasks. This is because the construction method is realistic and tailored for a data integration pipeline. We also highlight that GOBY is semantically-rich as compared to with other benchmarks. The most common labels are those related to events, such as location and organizer details. Meanwhile, typical benchmarks contain labels with low semantic information, for example the three most popular semantic types in one benchmark [53] are the generic labels `id`, `name` and `type`.

Further, because of the method of construction, GOBY has more realistic table structure. This can be seen in the average number of rows: 3 400, and columns: 20. This reflects the need to have a minimum amount of business information about each event. Prior benchmarks such as VizNet have an average of only 3 columns per dataset, while T2Dv2 has only 118 rows per table; both are unrealistic. The larger number of rows and columns reflects the reality of enterprise data.

While some of the underlying data from GOBY has been used in a prior data curation paper [97], we now curate and release the dataset as a benchmark for the first time.

GOBY is a private benchmark, according to the public-private taxonomy presented in section 4.1. In particular, while the domain doesn't require proprietary knowledge (events around town), the data is not from web tables but scrapped with human-programmed scripts from APIs, then labelled

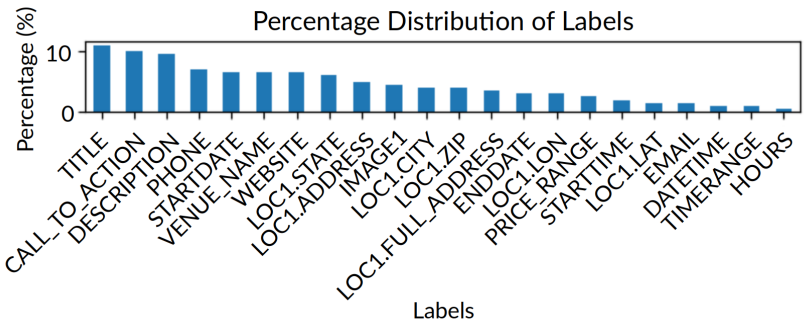


Figure 4.1: Semantic column label distribution in the GOBY dataset.

with private labels. To the best of our knowledge, GOBY is the first data integration benchmark derived from a private dataset.

GOBY does not contain any private personal identifiable information (PII). It also does not contain any data pertaining to individual users.

Download GOBY at `https://goby-benchmark.github.io/`.

4.3 Insights and Experiments

In this section, we explore the performance of large language models (LLMs) in the context of enterprise data, highlighting the limitations of previous models and benchmarks, as well as novel approaches we discovered for optimizing LLM performance. Through our experiments, we present key insights that not only build upon prior work but also introduce innovative techniques for integrating and analyzing enterprise data using LLMs. To backup these observations, we provide empirical evidence and detail in the design decisions made during the model development process, offering a clear reasoning for each step.

Experimental setup We conduct an experimental evaluation using the following large language models: OpenAI's `gpt-3.5-turbo` and Meta's `Llama2` with 13 billion parameters. Experiments utilizing grammar-constrained decoding were conducted using `Llama2` since OpenAI APIs do not expose the required raw token probabilities.

4.3.1 Enterprise performance gap

We begin by evaluating LLM approaches on the GOBY enterprise dataset in comparison to the public benchmark VizNet for the task of semantic column-type annotation. Overall, we find the outcome that LLMs perform considerably worse on GOBY compared to the public data benchmark.

The semantic column-type annotation task is a multi-class classification problem. We evaluate using the standard metrics precision P, recall R and their harmonic mean, the F_1 score. Precision is defined as ratio of true positives to the sum of true positives and false positives. Recall is defined as the ratio of true positives to the sum of true positives and false negatives. The F_1 score reflects a metric in which precision and recall are equally weighted. Because of the multi-class setting, the reported scores are macro-averaged. Additionally, we note that the LLM can decline to answer or provide an unparsable answer, lowering the recall but not precision.

Table 4.2: Performance discrepancies in LLM performance for column-type annotation on our enterprise benchmark Goby versus the public benchmark VizNet.

Metric	**Goby**	VizNet [62]	Difference
Precision P	77.1%	91.2%	14.1%
Recall R	70.0%	88.8%	18.8%
F_1 Score	0.71	0.89	0.18

The underlying approach for extracting semantic column-type predictions is the same as the CHORUS system of Chapter 3, to which we refer the reader for more details. In this experiment, the model is given access to the set of valid labels. We will remove this assumption in a later experiment.

Empirically, we find a gap between prior benchmarks and Goby of 0.18 F_1 points, 14.1 percentage points of precision and 18.8 precentage points of recall. This is shown in Table 4.2.

4.3.2 Iterative dictionary construction

The first step in our approach involves synthesizing a data dictionary and then generalizing it into an ontology. A Data dictionary is a list of possible data types and acceptable values, optionally with some metadata on their semantic meaning. Ontologies can be considered to be data dictionaries with a hierarchy.

The development of data dictionaries involve making many arbitrary distinctions. Label granularity is an example: the curator may choose to have the classes `street address`, `full address` and `PO box` or combine them under just one heading, `address`. Further, because of coordination failure between multiple labelers, redundant synonymous labels such as `wages` and `salary` can be present. As such, finding a perfect match for the dictionary created by a human curator is *not* the goal.

Now we start testing the ability of large language models to iteratively construct the class labels. In this experiment, we randomly sample from the Goby tables to learn the classes. The process is seeded with a small number of starting classes. We then consider a column at a time, presenting a sample of 5 unique values from that attribute. If the LLM predicts an existing class for the column, we continue onto the next column. Otherwise, we extract a new label for the column and add that to the pool of labels. The process is terminated when the quiescence condition is met: the observation of 5 tables sequentially, all of which do not require the creation of any new labels.

The evaluation metric here is *coverage*, the portion of ground-truth label for which there exists a synthesized label. We find that the LLM can obtain **70.5% coverage**. This is shown in Table 4.3. In the 29.5% of classes that are not recovered, 20% are due to incorrect granularity and 10% are semantically wrong. This motivates the next experiment to recover from granularity failures.

4.3.3 Building a Hierarchy

We address the challenge of label granularity by learning an ontology: a structure of classes. Starting with the data dictionary of the relevant classes, we aim at the goal of creating an ontology. We consider this ontology successful if it faithfully captures all ground-truth, human-labeled classes. We show an example in Figure 4.5. In section 4.3.3, we will show that such a synthesized ontology can capture 100% of the ground-truth labels and resolves the granularity issue.

First, we test the LLM's ability to transform its predicted classes into a hierarchy. We build the tree in an iterative approach, similar to the class learning prior. The main difference is that we

Table 4.3: Ability to reconstruct classes. This table of the percentage of classes were we able to infer, as compared to the number of classes in the ground truth labels.

True Label	Predicted Label
Exact Match (71%)	
LOC1.ADDRESS	Address
HOURS	OperatingHours
LOC1.LON	Geolocation
...	
Semantic Match (20%)	
STATE	Location
ZIP	Location
...	
No Match (9%)	
DESCRIPTION	Activity
...	

utilize batching to give the model enough context to select appropriate superclasses. We find this a resounding success: the model can produce superclasses that have 100% coverage of both the learned and ground truth classes. We show the learned ontology in Figure 4.5. We also show the mapping from ground truth labels to the ontology in Table 4.4.

Now that we have the ontology, integrating a hierarchical approach into LLM requires a number of architectural decisions that we experiment with in the next sub-section, section 4.3.4. As we will see, how the hierarchy is integrated into the LLM makes a large impact on the performance.

4.3.4 Encoding: Full context required

Once the ontology is learned, it must be encoded into the language model for prediction. We investigate several methods of integrating hierarchical information into LLMs. These included both architectural and semantic approaches. We consistently found that approaches that presented the full context to the LLMs outperformed those which attempted to feed only the relevant context.

Grammar-constrained decoding Large language models perform sampling to extract tokens from the conditional token probability distributions they output [48]. Common approaches include beam search, top-k, temperature and nucleus sampling. This architecture leads to the possibility of constraining the search space of the sampling to match some grammar [106]. Figure 4.2 shows the mechanism by which this approach works.

Listing 4.1 (Sample GBNF Grammar used for GCD)**.**

```
root ::= answer
answer ::= "Semantic-type::" property
property ::= event | location | contact | misc
event ::= "EVENT::" sub-event
sub-event ::= "TITLE" | "STARTDATE" | "DURATION" ...
location ::= "LOCATION::" sublocation
```

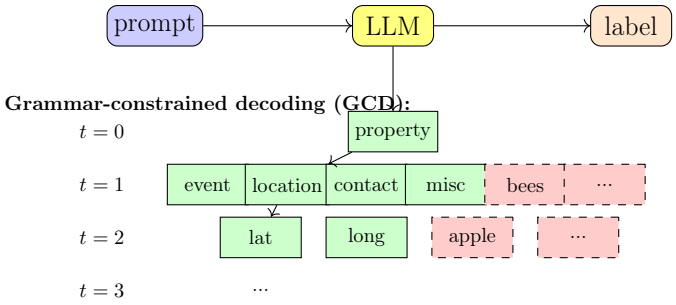


Figure 4.2: Mechanism of grammar-constrained decoding (GCD). GCD is applied to ensure the LLM complies with the ontology to find for given data instance a correct hierarchical-classes. Possible classes are constrained to the given dictionaries and relations are based on the previous layer. During decoding only valid token continuations compliant with the grammar (shown in green) are considered. Tokens outside the grammar (shown in red) are rejected.

```
sublocation ::= "LAT" | "LONG" | "ZIP" | "CITY" | "STATE" ...
contact ::= "CONTACT::" sub-contact
sub-contact ::= "WEBSITE" | "URL" | "PHONE" | "EMAIL" ...
meta ::= "META::" sub-misc
sub-meta ::=  "RATING" | "TOTAL_NUMBER_OF_RATINGS" ...
```

Step-by-step prompting Another method to encode the ontology is presenting a series of prompts within the same context. This is illustrated in figure 4.4. Starting at the root of the class tree, the model is presented with the first level of the ontology. Once a super-class is chosen, the children of that node are presented to the model in the next prompt. This process is repeated until the model selects a leaf node. This approach can be considered a chain-of-thought method.

Tree serialization The final approach we consider is serializing the class tree and providing it to the model. This involves traversing the tree in pre-order and enumerating each path from the root. This allows the model's attention mechanism to selectively attend to more valid paths in the tree. Surprisingly, as will will show shortly, this approach has the most robust performance.

Evaluation We consider a prediction correct if it matches the ground truth label in our mapping, or if the most-recent common ancestor (MRCA) is the direct parent of the prediction. We find the tree-serialization approach most effective, bringing total F_1 score to 0.85. Grammar-based decoding paradoxically results in a lower score F_1 0.66, while step-by-step approach achieves a modest lift of 0.05 F_1 points from the baseline of 0.71. This is in line with prior work that finds the GCD produces sequences that have overall low probability [85]. We note that the commonality between the two lower-performing approaches is that the LLM cannot examine the whole ontology: in the step-by-step prompts only one path is examined (no backtracking) while in the GCD-approach only nodes the beam search examines are considered. The approach that loads the whole ontology into the context performs best.

Legend: Instruction[I], Data sample[S], Metadata[M], Task-specific knowledge[K], Prefix[P]. The superscript letter repeats the colour code, so the figure can be read in greyscale.

```
You are a helpful data analysis assistant. Suggest a semantic type from these
classes. Reply with a single class.[I]
CLASSES:
PROPERTY::EVENT_INFO
PROPERTY::EVENT_INFO::TITLE
PROPERTY::EVENT_INFO::DURATION
...
PROPERTY::CONTACT_DETAILS
PROPERTY::CONTACT_DETAILS::PHONE_NUMBER
...[K]
TABLE: serialized_table
COL-NAME:‘col_name‘[M]
VALUES: sampled values[S]
SEMANTIC-TYPE:[P]
```

Figure 4.3: Tree-serialization approach. This figure shows the model prompt when serializing the ontology tree and presenting it to the model as one structure.

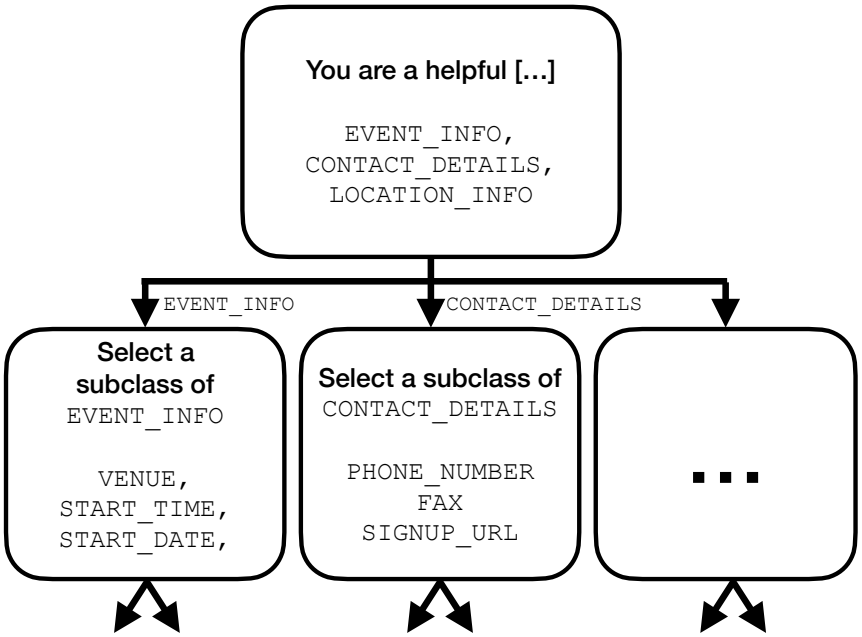


Figure 4.4: Step-by-step prompting. The model navigates a tree of potential prompts by making completions at each node. This figure shows the sequence of prompts used in this mode of operation, allowing the LLM to reason step-by-step. The prompt structure and table serialization are in line with those shown in Figure 4.3.

Table 4.4: Mapping the ground-truth classes into our ontology. This table shows that our ontology can accommodate all the ground-truth labels, resolving the granularity issue illustrated in Table 4.3. Overall, we achieve 100% coverage of the classes.

Learned Mapping	Ground Truth Label
PROPERTY::LOCATION::FULL_ADDR	LOC1.ADDRESS
	LOC1.STREET_ADDRESS
PROPERTY::CONTACT::URL	CALL_TO_ACTION_URL
	WEBSITE
	OTHER_URL
	VENUE_LINK
	MORE_INFO
...	

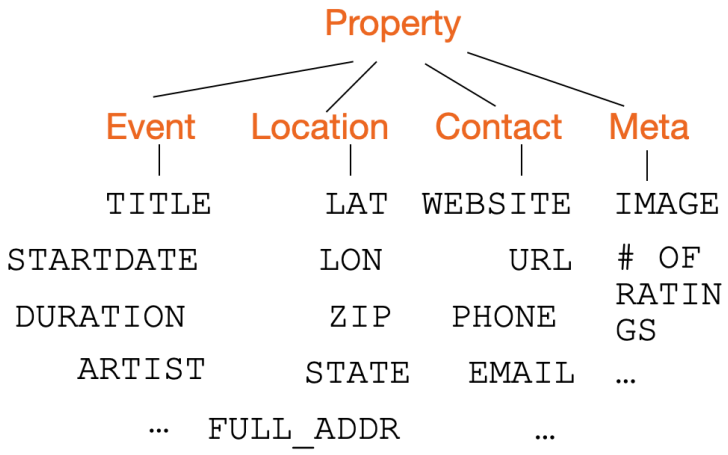


Figure 4.5: Example of the synthesized ontology. The base classes, in black, are first learned iteratively. The super-classes, in orange, and their relations are built in the ontology-learning step.

4.4 Discussion

Our findings support the hypothesis that enterprise data presents new challenges to LLMs not captured by public-data benchmarks. We find that a hierarchical approach allows for successfully learning labels and making predictions on enterprise data.

Dataset contamination or distribution shift? Two principal mechanisms may explain the observed performance gap. The *data contamination* theory posits that the LLMs have been trained on (portions of) test set of the benchmark dataset. This is because the massive corpuses they are trained on are composed largely of web data, which may include the downstream test datasets. Further, LLM have the demonstrated ability to "memorize" discrete sequences (such as individual addresses or phone numbers) and recall them on demand—albeit only when these sequences are duplicated many times in the corpus, at least 10-100 occurrences [60]. This contamination has been confirmed for a number of benchmarks. For instance, the MMLU (Massive Multitask Language Understanding) benchmark was also being tested and GPT-4 was able to accurately guess missing options in the benchmark test with 57% exact accuracy [24].

Another explanation is dataset shift. As explained earlier, this involves a change in the generative distribution underlying the training and runtime populations. These models also struggle on recall of tail entities, averaging 6-20% accuracy across all domains [100]. For tail entities in specialized domains, such as literature or academia, accuracy on tail entities drops to 3-5% range [100]. This is evidence against the LLM memorizing benchmark answers for these more specialized benchmarks. Other evidence in support of the data shift cause is recent work which tests LLM performance on data drawn from the same distribution as the benchmarks but produced after model training. On this sub-population which temporally cannot be subject to data contamination, the authors find minimal change in model performance [62].

Cost This also remains an elusive facet of LLM applications. Generally, the volume of "dark" enterprise data dwarfs that which is publicly available. Findings that LLMs are too cost prohibitive to run on public data are magnified in the enterprise setting, particularly due to extensive data duplication in the business setting. Effective table summarization becomes key: it is necessitated by the limited context window size of language models, which causes an inability to ingest whole tables. Current approaches for summarization are ad-hoc—a principled approach would be welcome. Scalability remains a unsolved challenge too. The superior performance of the (relatively context-consuming) tree serialization approach raises concerns.

Limitations The benchmark is one workload, from one industry, compiled at one point in time, and the gap is measured on one task. Whether the same shortfall—and the same remedy—appears in a clinical or financial schema is untested. The measurement itself is narrow in a second way: the 0.18 F_1 gap of Table 4.2 compares one LLM-based method against its score on one public benchmark, so the corresponding claim about representation-learning models rests on prior cross-dataset results rather than on anything measured here. The evaluation is also lenient by construction. A prediction counts as correct when its most-recent common ancestor with the ground-truth label is the label's direct parent, a one-level relaxation adopted because an ontology learned from data and one written by domain experts are unlikely to agree at leaf granularity; strict exact-match scores would be lower for every method. The mitigations were developed against Goby and evaluated on it, with no held-out enterprise dataset to confirm they generalize rather than fit. Finally, no released benchmark stays uncontaminated indefinitely. We distribute Goby behind a password so that it is not picked

up by automated crawlers, which is the reason to expect it to remain a clean test of parametric knowledge for some time, but nothing prevents a downstream user from placing a copy in a training corpus. The durable answer is a protocol—temporal hold-out, identifier scrambling—rather than any particular test set.

The insights gleaned in this work translate more widely into general data management problems: we have a work-in-progress on applying lessons learned in this approach to problem outside data integration, focusing on natural language to query translation in the enterprise.

4.5 Summary

Goby is the first data-integration benchmark built on real enterprise data. On it, both LLM- and ML-based methods for semantic column-type annotation lose roughly 0.10 F_1 points relative to their scores on public benchmarks such as VizNet—evaluating these systems on public data alone paints a false picture—and treating the ontology as something to be learned from data samples, then serialized in full, recovers most of that loss. The benchmark and its expert ontology are publicly released.

What the benchmark cannot settle is worth stating plainly. It is one workload, from one industry, on one task; whether the same gap and the same remedy appear in a clinical or financial schema is untested, and the mitigations were tuned against this data. The mechanism behind the gap also remains open. Section 4.4 weighs contamination against distribution shift and finds the evidence pointing toward shift—tail-entity recall is poor even on data that postdates training—but the two are not cleanly separable with the instruments available, and a benchmark that separated them would be a contribution in itself. Nor does one benchmark stay honest indefinitely: a released dataset is a dataset that can be trained on, which is an argument for protocols such as temporal hold-out and identifier scrambling rather than for any particular test set.

The finding that generalizes beyond the task concerns grounding. What closed the gap was not a better prompt or a larger model but the decision to derive the vocabulary from data rather than assume the model knew it. Chapter 5 carries that decision into query translation, where the vocabulary is hidden by construction; Chapter 6 develops it as this dissertation's organizing lesson.

Chapter 5

Query Blueprints: Factoring Structure from Vocabulary

An analyst wants to know which companies work in advertising. The question is easy to state and easy to check: given a candidate answer, anyone can say whether it is right. Answering it from a database is not one problem but two.

The first is a question of *structure*. Whatever the database looks like, the answer is a set of companies, selected by a condition on the industry they work in. The analyst knows this without knowing anything about the database: it follows from the question, not from the schema. The second is a question of *vocabulary*. This particular database calls a company something. It records an industry under some identifier. Whether that identifier is `industry`, `SIC_CD`, or `wdt:P452`, and whether the industry hangs off the company directly or two joins away, is a fact about the schema that no amount of thinking about the question will reveal.

Analysts reliably know the first and reliably do not know the second. Every natural-language-to-query system built to date nonetheless asks a single model to produce both at once, from a single English sentence, in one shot. This chapter argues that the coupling is the defect, and shows what can be built once the two are separated: the user states the structure exactly, and the system resolves the vocabulary by grounding it in the data. Before making that case, we develop the setting in which the rest of the chapter operates.

5.1 Querying a Knowledge Graph

A *knowledge graph* stores facts as labeled edges between entities. Figure 5.1 is a fragment of one: each node is an entity, such as the company WPP or the class *business*, and each edge asserts a single fact about a pair of them. An edge carries a *predicate*, the name of the relationship it asserts. The graph in the figure records nine facts, among them "WPP is an instance of business" and "Ogilvy's industry is advertising."

The representation is deliberately uniform: there are no tables, no columns, and no fixed record layout, only nodes and labeled edges. This uniformity is what makes a knowledge graph a convenient object of study here. A relational schema forces heterogeneity into two places at once—which table a fact lives in, and which column within it—whereas a graph forces it into one, the predicate on the edge. The problem does not become easier, but it becomes easier to isolate.

Now return to the analyst's question. On the graph of Figure 5.1, *which companies work in advertising?* has the answer $\{\text{WPP}, \text{Omnicom}\}$: Siemens is excluded because the only industry reachable from it is medical technology. A reader can compute that answer by inspection, without

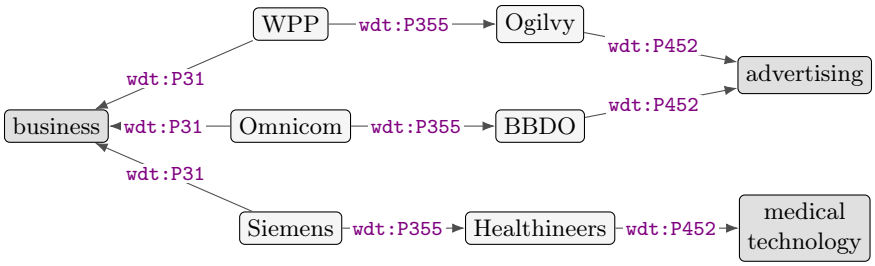


Figure 5.1: A fragment of a knowledge graph, in the style of WikiData. Nodes are entities; each labeled edge is one fact. The three predicates are `wdt:P31` (*instance of*), `wdt:P355` (*has subsidiary*), and `wdt:P452` (*industry*). Nothing in the graph itself reveals those glosses.

knowing what `wdt:P452` is called or what it means, simply by following edges.

A machine needs the same walk written down, and the language for writing it is SPARQL. A SPARQL query states a pattern of edges with variables in place of the unknown nodes, and returns every assignment of entities to variables that makes the pattern true of the graph:

```
SELECT ?x WHERE {
?x wdt:P31 wd:Q4830453 .
?x wdt:P355 ?z .
?z wdt:P452 wd:Q37038 .
}
```

Read declaratively, this says: find every ?x that is an instance of business, that has some subsidiary ?z, where that subsidiary's industry is advertising. Matched against Figure 5.1, it binds ?x to WPP (via ?z = Ogilvy) and to Omnicom (via ?z = BBDO), and to nothing else. That is exactly the answer computed by hand above. Throughout this chapter we use color to distinguish variables, NL fragments, entities and predicates, and literals.

Two features of that query deserve attention, because the rest of the chapter is about them. First, every symbol in it other than the variables is opaque. The analyst had to know that businesses are `wd:Q4830453`, that advertising is `wd:Q37038`, and that `wdt:P452` rather than any of WikiData's other 57,895 properties expresses "industry." None of this is derivable from the question. Second, the query is a three-edge path, but the question mentions only two things—a company and its industry. There is no edge from a parent company to an industry anywhere in this graph; the industry is recorded on the subsidiary. The natural-language phrase "works in advertising" does not correspond to a predicate at all. It corresponds to a *path*.

Why knowledge graphs. The setting this work is ultimately aimed at is the enterprise relational database, with its tens of thousands of opaquely named tables and columns. No such database is publicly available—that is precisely what makes it enterprise data—so it cannot serve as an object of open evaluation. Large knowledge graphs with comparably complex schemas can: WikiData exposes 57,896 predicates under identifiers that carry no meaning on their face, which reproduces the essential difficulty at a scale that can be measured and released. The techniques developed here are not specific to graphs; Section 5.6 carries them to the relational model and Section 5.7 evaluates them on the enterprise data of Chapter 4.

5.2 The Translation Problem

Schema heterogeneity is one of the principal challenges of querying structured data [26], and the scale of it is what rules out the obvious remedy of simply learning the schema. A typical deployment of Oracle eBusiness Suite or Microsoft Dynamics AX has over 100,000 attributes in its schema [13]; SAP ERP installations contain upwards of 90,000 tables with opaque abbreviated names [89]; the MIMIC-II clinical database requires a 76-page data dictionary just to document its schema [19]. No analyst can be expected to know such schemas in their entirety, yet writing a correct query demands exactly this knowledge. The challenge extends to LLM-based agents, which achieve only 22% accuracy on schemas with as few as 17 tables [37].

Given the analyst's question and a graph, then, the task is to produce the SPARQL query above. Four properties make it hard.

The vocabulary is unguessable, and models conceal that. A language model translating this question against WikiData often succeeds, but not by reasoning: it has read WikiData and remembers that `wdt:P452` is industry. Remove that memory—by scrambling the identifiers, as we do in Section 5.7.1, or by pointing the model at a proprietary schema it has never seen—and the performance collapses. We measure this collapse directly: four state-of-the-art agents fall below 10% accuracy on a scrambled schema. The model does not report the collapse, because a hallucinated predicate is syntactically indistinguishable from a correct one.

The translation is not structure-preserving. As Figure 5.1 shows, one natural-language relationship may be realized as a chain of predicates. A system that assumes a one-to-one correspondence between phrases and predicates—which is what schema matching and predicate lookup assume—cannot express the correct answer at all, no matter how good its similarity function. The search space is over *paths* through the schema, not over identifiers, and it grows exponentially in path length.

There is no training data, and there will not be. Supervised translation requires question–query pairs over the target schema. For a proprietary schema none exist; producing them requires the expert whose absence created the problem. Any method that must be trained per schema is disqualified at the outset.

Errors are silent. A query naming the wrong predicate does not fail. It returns rows—plausible ones, in the right shape, in the right number—that answer a different question than the one asked. In the relational experiments of Section 5.7.4, this is the dominant failure mode of the LLM baseline: 60% of its queries are structurally fluent and pointed at entirely wrong columns. Correctness here cannot be delegated to the user's eye.

5.3 Why Blueprints

These four properties narrow the design space considerably, and it is worth walking through what they exclude before presenting what we retain.

An **end-to-end LLM agent** is the default answer, and it fails on the first property: with the schema vocabulary absent from its parameters, it has nothing to fall back on, and by the fourth property it fails without saying so. **Retrieval-augmented generation** over the schema appears to fix this by supplying the vocabulary at query time, but retrieval is itself vocabulary-dependent—it

Query Blueprint

```
SELECT ?x WHERE {
  ?x ‘type‘ ‘company‘ .          // NL predicate
  ?x ‘industry‘ ‘advertising‘ .
}
```

Translated Query

```
SELECT ?x WHERE {
  ?x wdt:P31 wd:Q4830453 . // instance-of → business
  ?x wdt:P355 ?z2 .          // industry
  ?z2 wdt:P452 wd:Q37038 . // has-part → advertising
}
```

Figure 5.2: Query blueprint and its translated query. The translation replaced natural language strings with correct identifiers, and adjusted the query structure from 2 hops to 3 hops.

must match the user's words against identifiers that share no lexical overlap with them—and the model remains free to override retrieved evidence with parametric knowledge [34]. **Fine-tuning on the target schema** is precluded by the third property. **Classical schema matching** [73, 91] is exactly the right instinct—it grounds correspondences in instance data rather than in recall—but it maps element to element, and the second property says that the correspondence we need is element to path. **Query-by-example and keyword search** [2, 49, 76] do search over paths, and are the closest antecedent, but they place the entire burden on the data: the user supplies example answers or bare keywords, and the system must infer the query shape as well as the vocabulary. That discards information the user actually has.

The last of these is the telling one. Every approach in the list either demands schema knowledge the user lacks, or throws away the structural intent the user possesses. The design point that remains is the one that takes each from the party that has it: *structure from the user, stated exactly and never guessed at; vocabulary from the data, mined and never recalled.*

A *query blueprint* is that design point made concrete. It is a query in the host language—SPARQL, here—in which some predicates and constants are replaced by natural-language descriptions. The user writes the joins, filters, and projections in formal syntax, because they know what the query should compute; they write `‘industry‘` wherever they do not know what the schema calls a thing. Figure 5.2 shows the blueprint for the analyst's question alongside the query it must become. Note what the translation had to do: it resolved four natural-language fragments to identifiers, and it turned a two-edge blueprint into a three-edge query. The user was never asked for either.

This division has a further consequence worth stating. Because the structure is given rather than inferred, the system's output is checkable against the structure the user wrote—a translation either respects the requested shape or it does not—and the ambiguity that dominates natural-language benchmarks, where 41% of questions admit several non-equivalent correct queries [37], is resolved by construction rather than by guessing at intent.

Our system, BlueprintMiner, translates a blueprint via *blueprint mining*, an example-driven approach that requires no training data and works with any knowledge graph or relational database. It makes very limited use of LLMs—only for the initial example generation—and instead borrows from traditional database techniques such as schema matching [9, 73, 91] and keyword search over databases [2, 49, 87]. As shown in Figure 5.3, BlueprintMiner expands the blueprint with fresh variables, generates a few grounding examples for them, probes the database to extract virtual

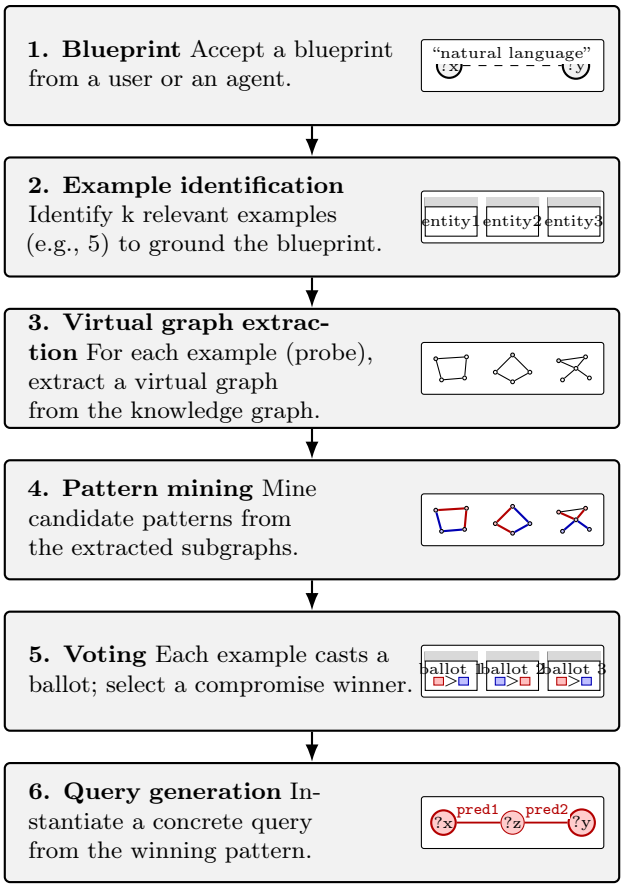


Figure 5.3: System overview: from input blueprint to concrete query via examples, virtual graph extraction, pattern mining, and voting.

subgraphs around each example, mines candidate query patterns via random-walk scoring, and finally selects a consensus translation via instant-runoff voting. The translation need not be isomorphic to the blueprint: a single natural-language edge may expand into a multi-hop path.

Contributions

1. We introduce *query blueprints*, a formalism that extends graph queries with natural language, factoring the query problem into structure (the user's responsibility) and semantics (the system's responsibility). Section 5.4.

2. We propose *blueprint mining*, an example-driven algorithm that discovers translations without training data. Section 5.5.

3. We construct WikiScramble, a benchmark of 218 query blueprints derived from real SPARQL queries, together with ground-truth answer sets established through cross-knowledge-graph entity alignment. Sections 5.7.1–5.7.2.

4. We evaluate BlueprintMiner on WikiScramble and the Goby relational benchmark, showing that it outperforms state-of-the-art LLM-based agents. Sections 5.7.3–5.7.4.

5.4 Query Blueprints

We fix a set of entities $\mathcal{E}$, a set of literals $\mathcal{C}$, and a set of predicate names $\mathcal{P}$. An example of an entity is `wd:Q4830453` ("business"), examples of literals are 726 or *'London'*, and an example of a predicate is `wdt:P31` ("instance-of"). We call $\mathcal{N} := \mathcal{E} \cup \mathcal{C}$ the set of *nodes*. In addition, we denote by $\mathcal{X}$ a set of variables, and by Σ^* the set of strings, such as natural language fragments. Whenever we consider a function $f : A \to B$, we will assume that f preserves the entities/predicates/literals: if $x \in A$ is an entity, predicate, or literal, then $f(x) = x$, otherwise $f(x)$ can be anything.

Background. A *knowledge graph* is a tuple $K = (V, E)$ where $V \subseteq \mathcal{N}$ is a finite set of nodes, and $E \subseteq V \times \mathcal{P} \times V$ is a finite set of labeled edges. We denote edges by $s \xrightarrow{p} o$, where $s, o \in V$, $p \in \mathcal{P}$. The graph of Figure 5.1 has nine such edges, among them WPP $\xrightarrow{\texttt{wdt:P355}}$ Ogilvy.

A *query* is a triple $Q = (V_Q, E_Q, \bar{x})$ where $V_Q \subseteq \mathcal{X} \cup \mathcal{N}$ is a finite set of variables and nodes, $E_Q \subseteq V_Q \times \mathcal{P} \times V_Q$ is a finite set of labeled edges, while $\bar{x} \subseteq V_Q$ is a tuple of variables called *output variables* or *head variables*. Given a knowledge graph K, a *match* of Q in K is a homomorphism $h : V_Q \to V$ that preserves edges: if $x \xrightarrow{p} y$ is an edge in E_Q, then $h(x) \xrightarrow{p} h(y)$ is an edge in E. The *answer* of the query Q on the knowledge graph K is the set $Q(K) = \{h(\bar{x}) \mid h \text{ is a match of } Q \text{ in } K\}$. The three-edge SPARQL query of Section 5.1 is such a Q, with $\bar{x} = (?x)$; it admits exactly two matches in Figure 5.1, one sending ?z to Ogilvy and one sending it to BBDO, so its answer is $\{\text{WPP}, \text{Omnicom}\}$.

In this chapter we are interested in large knowledge graphs with a complex schema. Our running example is based on WikiData, which has about 57,896 predicates (properties). No user can possibly know such a large schema.

A *chain* is an expression $x \xrightarrow{c} y$, where $x, y \in V_Q$ and $c = p_1 \cdot p_2 \cdots p_k \in P^+$ is a sequence of predicates. The chain is an abbreviation of a path $x \xrightarrow{p_1} z_1 \xrightarrow{p_2} \cdots \xrightarrow{p_k} y$, where the intermediate variables $z_1, \ldots$ do not occur anywhere else in the query. In Figure 5.1, the relationship an analyst would call "works in" is the chain $?x \xrightarrow{\texttt{wdt:P355}\cdot\texttt{wdt:P452}} ?y$, of length two; no single predicate expresses it.

Query Blueprints. A *query blueprint* B is defined like a query, but edge labels and constants may also be natural language (NL) descriptions. That is, $B = (V_B, E_B, \bar{x})$ where $V_B \subseteq \mathcal{X} \cup \mathcal{N} \cup \Sigma^*$ and $E_B \subseteq V_B \times (\mathcal{P} \cup \Sigma^*) \times V_B$. We assume that the strings in Σ^* occurring in the blueprint are natural language (NL) fragments. Notice that any query is also a query blueprint, but a blueprint is more general, since it allows NL fragments in place of nodes or predicates. The analyst's question of Section 5.1 becomes the two-edge blueprint of Figure 5.2, in which `'type'`, `'company'`, `'industry'`, and `'advertising'` are all NL fragments and only the shape is fixed.

Blueprint Translation. A *translation* of a blueprint $B = (V_B, E_B, \bar{x})$ is a query $Q = (V_Q, E_Q, \bar{x})$ such that $V_B \subseteq V_Q$ and for every edge $x \xrightarrow{p} y$ in the blueprint Q contains a (semantically equivalent) path $x \xrightarrow{c} y$. The translation must be *faithful*: Q must capture the intent expressed by the natural language descriptions in B, so that $Q(K)$ returns results that match the user's intent.

We leave the definition of a translation intentionally vague. We only require that the blueprint and its translation have the same number of output variables, and that they capture the same "intent". Notice that the translated query Q is not necessarily isomorphic to the blueprint B, because an edge $x \xrightarrow{p} y$ in the blueprint may be mapped to a path $x \xrightarrow{p_1} z_1 \xrightarrow{p_2} \cdots \xrightarrow{p_k} y$ in the query. This is not a corner case: it is what the blueprint edge $?x \xrightarrow{\texttt{'industry'}} \texttt{'advertising'}$ requires on the graph

of Figure 5.1, where the industry is recorded on the subsidiary and not on the company.

Syntax. We use the SPARQL syntax to denote queries, and use a simple extension of SPARQL for query blueprints. The extension adds two new production rules to SPARQL's EBNF grammar [46], consisting of a new alternative for predicates and for constants used in object positions that allow NL fragments wrapped in backquotes for predicates and constants respectively; Fig. 5.4 shows the extension for predicates; the extension for objects is similar. We use backquotes rather than quotes to avoid collision with SPARQL's regular quotes for constants: for example `` `New York` `` is an NL fragment while *'New York'* is a standard SPARQL constant. This minimal syntactic change enables any existing SPARQL parser to be extended with blueprint support, and any valid SPARQL query remains a valid blueprint.

Example 5.1. Figure 5.2 illustrates both a query and a query blueprint in our SPARQL-based syntax. For example, the expression `?x` `` `type` `` `` `company` `` in the blueprint represents the edge $?x \xrightarrow{\texttt{`type`}} \texttt{`company`}$, while the expression `?x wdt:P355 ?z2` in the query represents the edge $?x \xrightarrow{\texttt{wdt:P355}} ?z2$. Matched against the graph of Figure 5.1, the translated query binds `?x` to WPP and Omnicom, while the blueprint's two edges have become the query's three.

Extension to the Relational Model The definitions above are stated over knowledge graphs, for the reasons given in Section 5.1. They extend to relational databases with complex schemas; Sec. 5.6 gives the extension, and the relational experiments of Sec. 5.7 are conducted on Goby [63], one of the very few enterprise-like database benchmarks that is publicly available.

Restrictions on the query language We restrict both queries and blueprints to be acyclic and connected, i.e. the query graph is a tree. Also, we do not allow variables for predicate names, i.e. we do not allow $?x \xrightarrow{?p} ?y$. While we do allow predicates in query blueprints (e.g. regular expression matching on strings), we ignore them during the translation and simply copy them unchanged into the translated query.

5.5 Approach

Given a blueprint B, BlueprintMiner translates it into a query Q that returns the results intended by blueprint. The translation problem is very challenging when the database has a complex schema. For example, the related NL2SQL translation problem has achieved great success on academic benchmarks, such as Spider [117] or Bird [55, 88, 101, 114], but remains largely unsolved for enterprise-grade schemas [37], because these have large, complex schemas, often with opaque predicate names like `Column1`, `Column2`. BlueprintMiner targets precisely the settings where the schema is complex, like the 57,896 predicate names in WikiData, and is designed to not rely on the predicate names, which are often opaque. Instead, BlueprintMiner builds on traditional database techniques, such as schema matching [9, 73, 91] and keyword searches over databases [2, 49, 87], and makes only limited use of LLMs.

```
- Verb ::= VarOrIri | 'a'
+ Verb ::= VarOrIri | 'a' | NL_Predicate
+ NL_Predicate ::= '`' String '`'
```

Figure 5.4: The syntax of blueprints shown as a `diff` from the SPARQL syntax [46]; only the non-terminal `Verb` is affected.

```
SELECT ?x WHERE {
  ?x ‘type‘ ?organization .
  ?x ‘industry‘ ?industry .
  VALUES ?organization { ‘company‘ }
  VALUES ?industry { ‘advertising‘ }
}
```

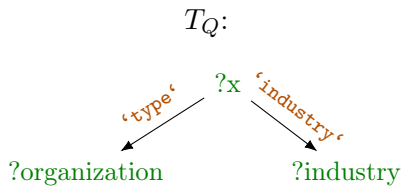


Figure 5.5: Query blueprint from Fig 5.2 is rewritten to introduce a fresh variable for each NL fragment. BlueprintMiner generates internal representation of the variables, but for readability we will use suggestive names: for example ?organization is the variable introduced for ‘company‘. The directed tree T_Q is shown on the right.

The steps of our translation are shown in Figure 5.3. BlueprintMiner starts by rewriting the user's input, then it identifies grounding examples for the nodes in the blueprint. Next, for each complete example it constructs a small subgraph of the KG connecting entities of that example. Next, for each subgraph it enumerates all candidate query patterns that would return the entities in that example, and scores the patterns using a score based on a random walk. Finally, it selects the winning pattern using voting, and converts the pattern into a SPARQL query. We describe the details in the rest of this section, and illustrate with our running example from Figure 5.2.

5.5.1 Blueprint input

BlueprintMiner reads a blueprint from the user, and rewrites it by introducing a fresh variable for each node that is given as an NL fragment; thus, in the rewritten blueprint each node is a variable or an entity. The connection between the fresh variable and the NL fragment is captured in a **VALUES** clause. Only NL fragments on nodes are affected by this step, those on predicates are unchanged. We also orient the query tree, and compute a topological order of its variables, $x_1, \ldots, x_n$, where, x_1 is the tree root. We denote by T_Q the ordered, directed tree representing the rewritten blueprint, and X_Q the set of its nodes, i.e. the variables of the blueprint.

Example 5.2. The query blueprint in Fig 5.2 is rewritten to the blueprint in Fig. 5.5. The topological order of the variables is ?x, ?organization, ?industry, and the directed, ordered tree T_Q is shown on the right of the figure.

5.5.2 Example Identification

Our goal is to convert the blueprint T_Q, whose edges are labeled with NL fragments $x \xrightarrow{\text{‘some NL‘}} y$, into a query where each edge is labeled with a chain of predicates $x \xrightarrow{p_1 \cdots p_k} y$ (and, thus, may have additional existential variables). The first step of the translation is to identify a set of examples, where an *example* is a mapping from X_Q to non-empty sets of entities in the KG: $h : X_Q \to 2^{\mathcal{E}}$. If each set $h(x)$ has size 1, then h represents a mapping from variables to entities (which is not yet a homomorphism, since it does not need to map edges to edges), but, in general, we need to allow a variable x to be mapped to a set of entities $h(x) \subseteq \mathcal{E}$, for reasons explained shortly. We generate k such examples ($k = 5$ in our implementation), as we describe in this subsection.

Correlated Example Generation. Large KGs exhibit internal heterogeneity: similar data items may be encoded in different ways, because they were ingested from different sources or at different times. For example, consider generating examples only for the ?*organization* variable. This occurs

in the edge $?x \xrightarrow{\text{`type`}} ?organization$, thus its intended meaning is that of the type of ?x, where "type" is represented in WikiData using the predicate "instanceOf" (`wdt:P31`). Some companies, like WPP[1] (`Q1318049`), are an instanceOf "Organization" (`Q43229`), while other companies like Interpublic Group[2] (`Q1501630`) are an instanceOf "Business Enterprise" (`Q4830453`). Thus, ?organization can be mapped either to `Q43229`, or to `Q4830453`, depending on the value of ?x. When generating examples for *?organization*, we must correlate them with the entity we associated to ?x, otherwise we risk producing entities that are unrelated in the KG.

BlueprintMiner generates *correlated* examples iteratively, starting from the root of T_Q, then conditioning the examples in $h(x_i)$ on those in $h(x_1), \ldots, h(x_{i-1})$. More precisely, we start by generating one example for x_1. The example is a string, which we link to the KG by using semantic-similarity-based entity matching. There are several entities in $\mathcal{E}$ that are candidate matches, and they form the first set of entities $h(x_1) \subseteq \mathcal{E}$. Next, for $i = 2, 3, \ldots, n$, we generate similarly an example for x_i, conditioned on the examples already generated for $x_1, x_2, \ldots, x_{i-1}$, and link it to a set of entities in the KG, to produce $h(x_i)$. We repeat this process k times, to generate k complete examples $h_1, \ldots, h_k$. To generate the individual examples, we support two mechanisms. The *generative approach* uses an LLM: this approach is best suited for queries over a general domain, such as WikiData, where an LLM is likely to produce good examples given the right prompt. The *retrieval-based approach* searches the KG directly using neighborhood embeddings. This is better suited for enterprise or domain-specific data, whose semantics may not be part of an LLM's training distribution. We describe both methods next.

Generative Approach. The generative approach relies on an LLM to produce examples. At each step i, we prompt the LLM with the entire blueprint, and with the previously generated examples for $x_0, \ldots, x_{i-1}$, and ask it to generate an example for x_i. The LLM responds with a string, which we link to the KG entities using the full-text search API (Wikidata's `wbsearchentities`, backed by OpenSearch).

Example 5.3. The example generation for our running example is shown in Figure 5.6. We ask the LLM to provide 5 examples for ?x, and include the entire blueprint in the prompt. The LLM responds with the first column in Figure 5.6. Consider the first string, "WPP plc". Using the full-text search of WikiData, we find a unique corresponding entity, thus we set $h_1(?x) := \{\texttt{Q1318049}\}$, whose standardized name in WikiData is "WPP". Next, we follow the edge $?x \xrightarrow{\text{`type`}} ?organization$ in T_Q and ask the LLM again for an example, prompting it with the blueprint, the string "WPP", and the string `'type'`: the LLM responds with "company". We link this string to WikiData and find two matches: `Q6881511` ("enterprise") and `Q783794` ("company"), thus $h_1(?organization) = \{\texttt{Q6881511}, \texttt{Q783794}\}$. Finally, we repeat this process for the edge $?x \xrightarrow{\text{`industry`}} ?industry$. The complete first example is:

$h_1 =$

?x	*?organization*	*?industry*
`Q1318049`	`Q6881511`, `Q783794`	`Q37038`

We repeat this for the other four examples of ?x. All 5 examples $h_1, \ldots, h_5$ are shown in Figure 5.6

Retrieval-Based Approach. The LLM-based approach will no longer work for completely proprietary enterprise databases, because they use unfamiliar concepts that will not occur in the

[1]Wire and Plastic Products plc is the largest advertising company in the world.
[2]An American advertising company based in New York City.

	?x		⇒	?organization		⇒	?industry	
	string	entity		string	entity		string	entity
e_1	*"WPP plc"*	Q1318049 (WPP)		*"company"*	Q6881511 (enterprise) Q783794 (company)		*"advertising industry"*	Q37038 (advertising)
e_2	*"Omnicom Group"*	Q1519267 (Omnicom)		*"company"*	Q6881511 (enterprise) Q4830453 (business) Q891723 (public co.)		*"advertising industry"*	Q37038 (advertising)
e_3	*"Publicis Groupe"*	Q1537378 (Publicis)	⇒	*"company"*	Q6881511 (enterprise) Q783794 (company)	⇒	*"advertising industry"*	Q37038 (advertising) Q23700481 (adv. industry)
e_4	*"Interpublic Group"*	Q1501630 (Interpublic)		*"company"*	Q6881511 (enterprise) Q4830453 (business)		*"advertising industry"*	Q37038 (advertising)
e_5	*"Dentsu"*	Q1189945 (Dentsu)		*"company"*	Q4830453 (business) Q6881511 (enterprise) Q783794 (company)		*"advertising industry"*	Q37038 (advertising)

Figure 5.6: Correlated example generation for the advertising blueprint. Pairs are generated one variable at a time, following the edges of T_Q. First, the LLM generates five strings for ?x (*"WPP plc"*, *"Omnicom Group"*, ...). Next we link each of them to a KG entity (e.g., Q1318049). Then, conditioned on each binding of ?x, the LLM generates a string for ?organization and we link it to one or more KG entities that are also reachable from ?x (e.g., Q6881511 and Q783794 for WPP). Finally, the same process is repeated for ?industry, again conditioned on ?x. This conditioning is essential: different ?x entities connect to different organization types in the KG, so generating pairs jointly ensures the linked entities are mutually reachable.

LLM-training. Our second example generation method is a retrieval-based approach that bypasses both LLM generation and entity linking by searching the KG directly. For each NL edge $x \xrightarrow{u} y$ in the blueprint, we embed entity neighborhoods in the KG and search for entities whose local graph structure semantically matches the NL description u. For each candidate entity a, we embed a's one-hop neighborhood (the set of predicates and neighbors connected to a) and compare it to an embedding of the NL description. Entities with the highest similarity become grounding examples directly.

For example, given a general NL edge $x \xrightarrow{\texttt{`industry`}} y$, this approach finds entity pairs (a, b) where a's neighborhood contains connections semantically similar to "industry" leading to b. When the target is a constrained variable, as in ?x `industry` ?industry with ?industry = `advertising`, the search is further constrained to entities whose neighborhoods include connections to entities similar to "Advertising," surfacing companies like Q1318049 (WPP) or Q1519267 (Omnicom) without an explicit generation step. This approach requires precomputed embeddings for all values in the database.

In summary, at the end of the Example Identification Step we have a set of k examples $H = \{h_1, \ldots, h_k\}$, where each example maps all query variables to non-empty sets of entities in the KG.

5.5.3 Virtual Graph Extraction

Next, BlueprintMiner computes for each example h a *virtual graph* G_h, which is a multigraph whose nodes are the example's entities $\bigcup_i h(x_i)$. Each edge (a, b) in G_h correspond to an edge (x, y) in the query tree T_Q, where $a \in h(x), b \in h(y)$. More precisely, G_h contains an edge $a \xrightarrow{c} b$ if the knowledge graph contains a path $a \xrightarrow{p_1 \cdot p_2 \cdots p_\ell} b$ labeled with the chain $c = p_1 \cdot p_2 \cdots p_\ell$. Intuitively, the chain c represents one candidate for the edge $x \xrightarrow{c} y$ to be included in the translated query. We compute the graph G_h by finding all paths of length $\ell \leq L$ in KG that link pairs of entities

$a, b \in \bigcup_i h(x_i)$, where $L = 3$ in our implementation. We also allow the path to traverse edges in the reverse direction, and write p^{-1} for the corresponding predicate. Notice that the nodes in G_h are in 1-1 correspondence with example's entities $\bigcup h(x_i)$, and do not include the intermediate nodes of the chains $p_1 \cdot p_2 \cdots p_\ell$. In particular, if two path between a and b have the same labels, then we record them as a single edge in G_h; this is not a subgraph of the KG, but a compressed summary of the paths connecting entity pairs. If G_h is disconnected, then we discard the example h.

Example 5.4. Figure 5.7 shows the virtual graphs for all five examples in Example 5.3. In each graph, the colored nodes correspond to the query variables, while the gray nodes are intermediate nodes on the predicate chains connecting them, and are not considered as part of the graph. WPP's virtual graph contains 4 distinct predicate chains; Omnicom's contains 10; etc. Across all five examples, 16 unique combinations of predicate chains are generated.

We explain some details of the first graph, which corresponds to h_1 of Example 5.3. It has 4 nodes, corresponding to the 4 entities occurring in h_1, and WikiData contains four chains of length ≤ 3 connecting these entities. Consider, for example the edge $?x \xrightarrow{\text{`type`}} ?organization$ in the blueprint. This corresponds to three chains from Q1318049 (WPP) to either Q6881511 (Enterprise) or Q783794 (Company):[3]

$$\texttt{Q1318049} \xrightarrow{\texttt{P31}(\text{instanceOf})} \texttt{Q6881511}$$

$$\texttt{Q1318049} \xrightarrow{\texttt{P355}(\text{has subsidiary})} \xrightarrow{\texttt{P31}(\text{instanceOf})} \texttt{Q783794}$$

$$\texttt{Q1318049} \xrightarrow{\texttt{P31}(\text{instanceOf})} \xrightarrow{\texttt{P31}(\text{said to be the same})} \texttt{Q783794}$$

Thus, while WPP is an instanceOf Enterprise, it is also connected to Company, because it is a subsidiary (of Finsbury Glover Hering, Q108564332), which is an instanceOf Company. Moreover, WPP is also an instanceOf Business Q4830452, which is "said to be the same as" Company.

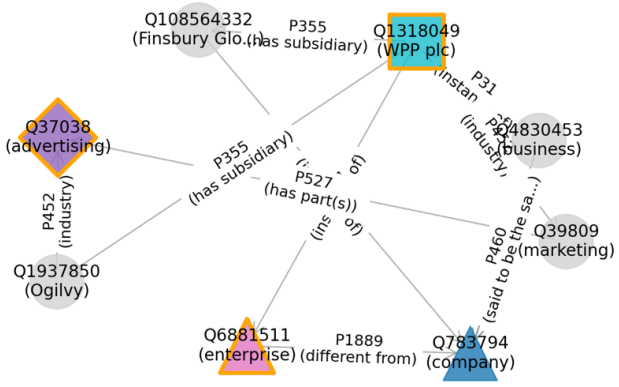
This example highlights the challenge in translating the blueprint. The translated query must include an edge $?x \xrightarrow{c} ?organization$, where c occurs in at least one of the five examples, and there are many candidates for c. Moreover, our choice for c must correlate with our choice for the chain c' labeling $?x \xrightarrow{c'} ?industry$, further increasing the difficulty.

5.5.4 Pattern Mining

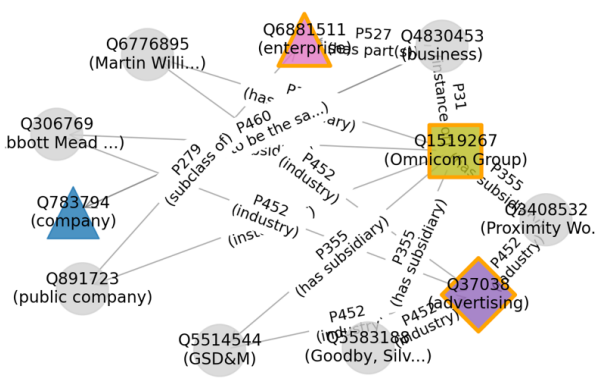
A *query pattern* is a subtree t of a virtual graph G_h satisfying the following: for every edge (x, y) in the query tree T_Q, the pattern t has exactly one edge $a \xrightarrow{c} b$, where $a \in h(x), b \in h(y)$. The query pattern is a complete candidate for our translated query Q. However, each graph G_h may contain multiple query patterns, and we have several graphs G_h: in this section we describe how to choose the query pattern that forms our translated query.

A naive approach to selecting a pattern is to score independently each edge in G_h, where higher scores are better, then compute a Steiner tree, i.e. a pattern with maximum weight. This was our first implementation, however, we observed a failure mode: the best predicate chain c for $x \xrightarrow{c} y$ may not join with the best chain c' for $x \xrightarrow{c'} z$, because the first chain corresponds to a path from $a \in h(x)$ to $b \in h(y)$, while the second corresponds to a path $a' \in h(x)$ to $c \in h(z)$, with $a \neq a'$, and no entity in $h(x)$ has both chains c and c'. Each chain may score well in isolation, but the

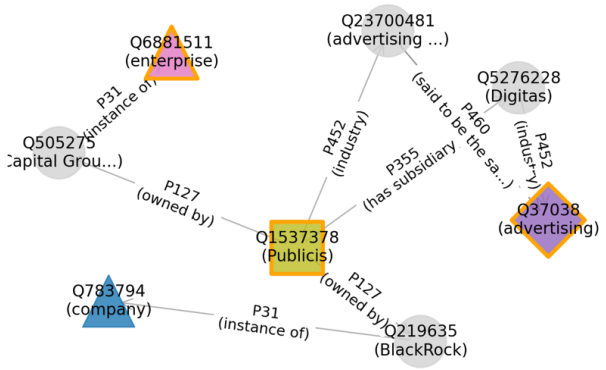
[3]The figures are automatically generated by BlueprintMiner and some labels may be difficult to read; this is the reason we expand them here.



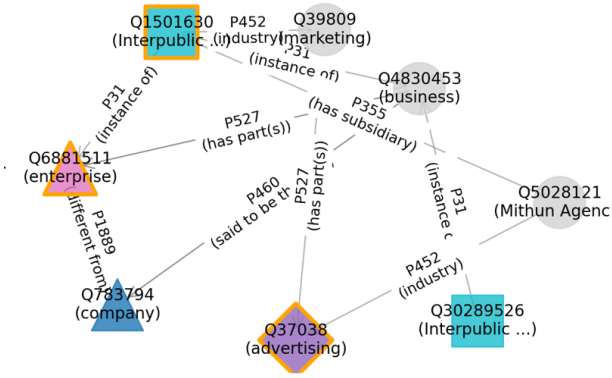
(a) h_1: WPP



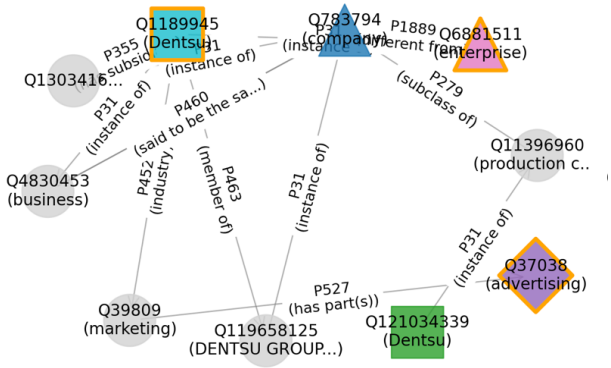
(b) h_2: Omnicom



(c) h_3: Publicis



(d) h_4: Interpublic

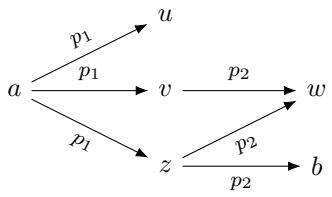


(e) h_5: Dentsu

Figure 5.7: Extracted subgraphs for the five advertising blueprint examples, showing individual predicates and intermediate nodes. The virtual graphs used for pattern mining retain only the predicate chains (sequences of edge labels) between example entities, discarding intermediate nodes.

combination produces patterns that matched no examples and do not exist in the KG. Scoring complete patterns eliminates this problem by ensuring that every candidate is grounded in at least one example.

Chain Scoring via Random Walks. We score each candidate chain using a random walk model. Consider a directed edge (x, y) in the tree T_Q, and a chain $a \xrightarrow{p_1 \cdots p_\ell} b$ connecting $a \in h(x)$ with $b \in h(y)$. We define $P(a \longrightarrow\!\!\!\!\!\rightarrow b \mid p_1 \cdots p_\ell)$ to be the probability that a random walk in the KG starting at a and following edges labeled $p_1, p_2, \ldots, p_\ell$ arrives at b. At each step, the walk chooses uniformly among all outgoing edges with the required label. A subtlety of this definition is that the denominator of the probability must include all attempted path from a, even those that get stuck in the middle. For example, assuming a tiny knowledge graph shown here:



we have $P(a \longrightarrow\!\!\!\!\!\rightarrow b \mid p_1 \cdot p_2) = \frac{1}{4}$, because there are 4 path from a, and only one reaches b. We have expressed this computation as a single SPARQL query that counts the total number of paths from a, following the predicate sequence, and the number that terminate at b, and were able to compute the pattern's probability in 1-2s. This scoring naturally penalizes chains that are too general.

Example 5.5. Consider the following blueprint, asking for all Pope successors:

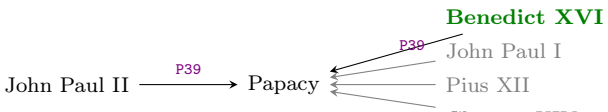
```
SELECT ?x ?y WHERE {
  ?x ‘successor‘ ?y .
  ?x ‘is a‘ ‘Pope‘ .
  ?y ‘is a‘ ‘Pope‘ .
}
```

One of the examples generated by the LLM (as per Sec. 5.5.2) binds correctly ?x to John Paul II (Q989) and ?y his successor Benedict XVI (Q2494). Our task to find and rank the predicate, or chain of predicates that connects them in the KG, i.e. find candidates for translating $?x \xrightarrow{\text{‘successor‘}} ?y$: this is illustrated in Fig. 5.8. WikiData has no direct link between John Paul II and Benedict XVI: while a ‘successor‘ predicate exists in WikiData (P156), it does not connect our two entities. A path of length 2 connecting them is P39 $\cdot$ P39^{-1} (position held $\cdot$ position held^{-1}), but this is not a correct translation of ‘successor‘, because any two popes are connected by this path, since both held the position of Papacy. The random walk correctly assigns a very low score to this path, namely $\frac{1}{267}$, as can be seen in Fig. 5.8 (b). The correct translation of ‘successor‘ is obtained from another path of length 2 connecting them, namely P1365^{-1} $\cdot$ P39^{-1} (replaces^{-1} $\cdot$ position held^{-1}), shown in Fig. 5.8 (c). The random walk model correctly scores c_3 as best, with a probability of 1.0.

During this step of the translation, we proceed as follows. For each virtual graph G_h, we enumerate all its tree patterns t. Recall that t has exactly one edge $a \xrightarrow{c} b$ for every tree edge $(x, y) \in T_Q$, where $a \in h(x), b \in h(y)$. We compute the score $s(t)$ of t as the product of all probabilities $P(a \longrightarrow\!\!\!\!\!\rightarrow b|c)$. Thus, each example h will be associated with a ranked list of patterns that occur in G_h.

$P(\text{JP II} \to \text{Ben. XVI} \mid c_1) = 0$

(a) $c_1 = \texttt{P156}$ (*successor*): although a valid predicate, and a perfect semantic match to the blueprint, no edge labeled P156 connects the two examples ($p = 0$).



$P(\text{JP II} \to \text{Ben. XVI} \mid c_2) = \frac{1}{267}$

(b) $c_2 = \texttt{P39} \cdot \texttt{P39}^{-1}$ (*position held* · *position held*$^{-1}$): diffuses over all 267 popes ($p = 1/267$).

$P(\text{JP II} \to \text{Ben. XVI} \mid c_3) = 1.0$

(c) $c_3 = \texttt{P1365}^{-1} \cdot \texttt{P39}^{-1}$ (*replaces*$^{-1}$ · *position held*$^{-1}$): uniquely identifies the successor ($p = 1.0$).

Figure 5.8: Random walk scoring for the blueprint edge $?x \xrightarrow{\texttt{`successor`}} ?y$, grounded on John Paul II ($?x$) and Benedict XVI ($?y$).

Example 5.6. For the advertising blueprint, one high-scoring pattern assigns `wdt:P31` to `'type'` → ?organization and `wdt:P452` to `'industry'` → ?industry. A competing pattern might use the two-hop chain $\texttt{P344} \cdot \texttt{P452}$ for the industry edge. Each example produces a ranked list of such patterns, ready for reconciliation.

5.5.5 Voting

At this point we select a single winning pattern among all patterns occurring in all virtual graphs G_h. First, we modify each pattern t by replacing the entities that form its nodes with variables X_Q of the query blueprint; thus, an edge $a \xrightarrow{c} b$ in the pattern becomes $x \xrightarrow{c} y$, where $a \in h(x), b \in h(y)$ and $x, y \in X_Q$. Each pattern becomes a complete candidate for the translated query. Let $T = \{t_1, \ldots, t_m\}$ denote the set of all candidate tree patterns across all examples: notice that the same pattern may occur in multiple examples, $G_{h_1}, G_{h_2}, \ldots$ For each example, h_i, let $s_i(t)$ denote the score (probability) of tree t under example i. In general, we have $s_i(t) \neq s_j(t)$, because examples h_i and h_j use different entities, and the random walk probabilities differ. If tree t does not appear in example i's subgraph, we assign $s_i(t) = 0$. At this point we need to select a single winning pattern t^*, based on its scores in all examples, $s_1(t^*), \ldots, s_k(t^*)$. For that purpose, we use voting: each example is a voter, and the patterns are the candidates.

A simple majority vote is undesirable. In practice, there are hundreds of candidate trees, and the very few that occur in *all* examples tend to be very general patterns—they achieve perfect recall by connecting all examples, but have very low precision because they also connect many irrelevant entities. These are precisely the diffuse chains we wish to avoid.

Similarly, we do not want any single example to have veto power. A particular example may be defective—perhaps its grounding entities were incorrectly linked, or the relevant path is missing from our bounded subgraph extraction. Requiring unanimous support would allow one bad example to eliminate the correct translation.

Consensus-Based Selection. Our desiderata align with consensus-based voting methods from social choice theory. These methods select candidates that appeal broadly to all voters, without placing undue weight on any single voter's strong preference or privileging polarizing candidates. Formally, let each example $i \in \{1, \ldots, k\}$ be a voter that submits a ballot $s_i : T \to \mathbb{R}$ ranking candidates by score. A consensus method f aggregates these ballots to select a winning candidate $t^* = f(s_1, \ldots, s_k)$.

In our setting:

Voters: the k examples

Candidates: all tree patterns from all examples

Ballots: voter i ranks the patterns by its scoring function s_i

Consensus methods satisfy desirable properties: they are not dictatorial (no single voter determines the outcome), they resist strategic manipulation, and they favor candidates with broad support over polarizing candidates with extreme scores from a subset of voters.

Instant Runoff Voting. We adopt Instant Runoff Voting (IRV) [21] as our aggregation method. IRV proceeds in rounds: in each round, every voter's ballot contributes one vote to their highest-ranked candidate that has not yet been eliminated; the candidate with the fewest votes is then eliminated. This continues until one candidate remains.

IRV offers a practical balance between consensus properties and computational efficiency. While Condorcet methods [20] (such as Schulze or ranked-pairs) have theoretical advantages—they select a candidate that would win pairwise against every other candidate, when one exists—their computational cost is higher: $O(m^2)$ pairwise comparisons for basic Condorcet, and $O(m^3)$ for methods like Schulze that resolve cycles, where $m = |T|$ is the number of tree patterns. In our setting, it is not unusual to encounter over 500 candidate trees, making $m^3 > 10^8$ operations. IRV, by contrast, runs in $O(m \cdot k)$ time where k is the number of examples (voters), making it practical for large candidate sets.

IRV has known disadvantages, such as by the Gibbard-Satterthwaite theorem [41] vulnerability to *strategic voting*—voters misrepresenting their true preferences to manipulate the outcome. However, the disadvantages of IRV do not apply in our setting: the voters are example subgraphs whose rankings we compute directly, and so preference falsification is impossible. We can guarantee that all ballots honestly reflect each example's scores.

5.5.6 Query Generation

The winning tree pattern t^* from voting represents our translation (Q, σ), as defined in Section 5.4. We simply expand each edge $x \overset{p_1 \cdot p_2 \cdots}{\longrightarrow\!\!\!\!\!\rightarrow} y$ in t^* into a set of edges $x \xrightarrow{p_1} z_1$, $z_1 \xrightarrow{p_2} z_2$, …, $z_{k-1} \xrightarrow{p_k} y$.

5.5.7 Extensions

BlueprintMiner allows the query blueprint to include other constructs, beyond the definition in Sec. 5.4: a **SELECT** clause, **FILTER** expressions, an **ORDER BY** clause, **GROUP BY** and aggregate functions, etc. These constructs are not accounted for by the translation, but simply copied to the output query. Any occurrence of an NL term is replaced with its translation. Our translation will introduce intermediate variables during chain expansion—these will be confined to the **WHERE** clause. Since we preserved all variables from the blueprint query, this ensures that the

translated query has the same semantics of the original blueprint, including aggregations, sorting, and filtering logic.

5.6 Relational Extension

We extend BlueprintMiner to relational databases via a standard relational-to-KG mapping. Given a relational database D, we construct a knowledge graph K_D as follows: each row r in a table T becomes an entity e_r, with a type edge $e_r \xrightarrow{\text{type}} T$. For each attribute p of T we add an edge $e_r \xrightarrow{p} v$, where v is the value of $r.p$; if p is a foreign key to a table T', then the edge is $e_r \xrightarrow{p} e_{r'}$, where r' is the record in T' referenced by $r.p$. Given this mapping, a SQL join path consisting of n key/foreign-key joins becomes an n-hop chain in K_D—exactly the multi-hop patterns that blueprint translation already discovers (Sec. 5.5).

With this view, the entire blueprint pipeline applies to relational databases without modification: we load the relational data, expose it as K_D via a SPARQL interface, and run blueprint mining as described in Section 5.5. We evaluate this in Section 5.7.4 on the Goby benchmark [63]. Because enterprise schemas are private and unlikely to appear in LLM training data, we use the retrieval-based example identification method (Sec. 5.5.2) rather than the generative approach.

5.7 Experiments

We evaluate BlueprintMiner on two benchmarks with deliberately opaque schemas: WikiScramble, a knowledge graph whose predicate and entity IDs have been scrambled to remove LLM overfitting to public KG vocabularies (Section 5.7.1), and Goby, a private enterprise relational dataset (Section 5.7.2). Against these benchmarks, we address the following questions.

- Does BlueprintMiner translate blueprints end-to-end on schemas unseen during pretraining, and how does it compare against state-of-the-art LLM agents? We answer this for WikiScramble in Section 5.7.3 and for Goby in Section 5.7.4.
- What does a BlueprintMiner translation look like in practice—is it a one-to-one predicate renaming, or structurally different from the input blueprint? (Section 5.7.3)
- Are the core techniques in BlueprintMiner—virtual graph extraction and instant runoff voting—empirically justified? (Section 5.7.3)
- Why do agentic baselines collapse on opaque schemas, and does the failure reflect overfitting to public schema identifiers? (Section 5.7.3)
- Do the advantages of BlueprintMiner carry over from the knowledge graph setting to relational data, as tested on Goby? (Section 5.7.4)

Table 5.1 shows the datasets used in the experiments.

5.7.1 WikiScramble Benchmark Design

In the natural use case, a user writes a blueprint against a single knowledge graph and the system translates it to that KG's vocabulary. However, this setting lacks ground truth: without knowing the user's precise intent, we cannot measure whether a translation is correct. In order to test our system, we had to create a benchmark of query blueprints on a large KG, where we had the ground truth.

Table 5.1: Datasets used in evaluation.

Dataset	Type	Entities	Predicates / Cols	Triples / Rows
WikiScramble	KG	115M	57,896	1.6B
DBpedia	KG	6.6M	2,813	438M
Goby	Relational	—	18,762	5.8M

Cross-KG Translation as Proxy. We built our benchmark by first collecting a large number of SPARQL queries over one knowledge graph, called the *source*, and translating each query into a blueprint on a different, unrelated knowledge graph, called the *target*, which has sufficient content overlap with the source. The input SPARQL query represents the user's intent, namely what entities and relationships they want, and the answers from the source KG represents the ground truth. But, when asked over the target KG, it faces an unfamiliar vocabulary that must be discovered by our translation. A translation is correct if the target query returns semantically equivalent results to the source query.

More precisely, we collected from GitHub a workload of real SPARQL queries over DBpedia (the source), which we converted into query blueprints. The benchmark asks for these blueprints to be answered on Wikidata (the target). This proxy is valid because DBpedia and Wikidata cover overlapping real-world entities but use completely different schemas (DBpedia has 2,813 predicates while Wikidata has 57,896, see Table 5.1), and the representation of knowledge is quite different (a single predicate in one KG often corresponds to several hops in the other KG) . The schema mismatch simulates the vocabulary gap that users face when querying an unfamiliar KG.

Query Collection. We collected real SPARQL queries from GitHub repositories that interact with DBpedia. Unlike synthetic queries generated by random walks over the knowledge graph, these queries reflect actual developer information needs and exhibit realistic patterns of predicate usage and entity selection.

We built a scraper using the GitHub Search API (inspired by [52]) to identify repositories containing DBpedia queries. The scraper searched for files containing DBpedia ontology prefixes (`dbo:`, `dbp:`, and `dbr:`), cloned the 4,131 identified repositories, and extracted the candidate files. Since a single file may contain multiple SPARQL queries—for example, a test suite or a configuration file with several query definitions—we used regular expressions to extract individual query candidates from each file, yielding 11,667 candidate query strings. Many of these were incomplete fragments or query templates containing placeholder variables intended to be filled in programmatically.

To filter to valid, executable queries, we parsed each candidate using a SPARQL parser. Of the 11,667 candidates, 3,425 were syntactically valid. We discarded the rest, including query templates containing unbound placeholder variables (e.g., `$variable` or `{parameter}`) that would not execute against a live endpoint.

Finally, we executed the valid queries against an up-to-date instance of DBpedia. We excluded queries that resulted in runtime errors (e.g., calls to unsupported functions), queries that returned no results (indicating the data may have changed since the query was written), and queries that timed out after two minutes (indicating excessive complexity or inefficient patterns). The remaining queries form our benchmark corpus. Figure 5.9 summarizes this pipeline.

Blueprint Synthesis. Given our corpus of DBpedia queries, we must generate corresponding query blueprints that capture the semantic intent of each query without relying on DBpedia-specific

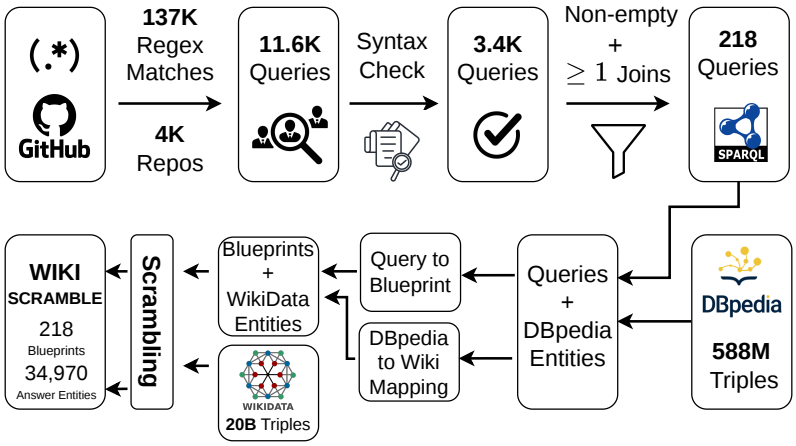


Figure 5.9: Benchmark creation. WikiScramble is built from three underlying sources: Github, DBpedia and WikiData.

vocabulary. We achieve this by systematically replacing the predicates and constants in each query with natural language descriptions derived from their English labels available in DBpedia.

For each predicate in the query (e.g., `dbo:birthPlace`), we retrieve its `rdfs:label` from the DBpedia ontology and substitute the predicate URI with this human-readable string. Similarly, for each constant entity (e.g., `dbr:Berlin`), we retrieve its English label and replace the URI accordingly. This transformation produces query blueprints that are *structurally isomorphic* to the original DBpedia queries—they preserve the exact graph pattern, variable bindings, and query form—but contain no fixed predicates or entity URIs, only natural language descriptions. For example, the DBpedia SPARQL query **SELECT** ?x **WHERE** { ?x `a dbo:Company` . ?x `dbo:industry dbr:Advertising` . } becomes our blueprint example in Fig. 5.2.

As long as the query blueprints are evaluated against a knowledge graph with a different schema, the translation process must map the natural language descriptions to the new schema's vocabulary. The translated query must have different structure and different URIs from the original.

Ground Truth Results and Entity Mapping. For each query in our corpus, we execute it against DBpedia to obtain the ground truth result set and extract the entity URIs that comprise the correct answer. We then map each DBpedia entity to Wikidata using `owl:sameAs` links that connect equivalent entities across the two knowledge graphs. Our benchmark consists of the resulting set of (blueprint, Wikidata-answer) pairs.

Schema Scrambling. A central feature of WikiScramble is that the Wikidata schema is made *genuinely* opaque. Although Wikidata predicate identifiers such as `P31` are syntactically opaque to a human reader, LLMs are trained on public datasets and have internalized the correspondence between these identifiers and their meanings—so that `P31` effectively means "instance of" to a pretrained LLM even when encountered without any label. We designed WikiScramble with this hypothesis in mind. To remove this potential shortcut, we apply a randomized bijective mapping over the predicate ID space: each predicate identifier is replaced by a different, randomly assigned Wikidata predicate ID, while preserving the human-readable `rdfs:label` annotations. For example, `wdt:P31` ("instance of") is remapped to `wdt:P9837`; the English label "instance of" is retained for `wd:P9837`

and remains accessible both via `rdfs:label` in SPARQL (e.g., `wd:P9837 rdfs:label` *'instance of'@en*) and via the semantic search tool over predicates. Critically, all tools used to explore the knowledge graph—including entity lookup and semantic search over predicates—are updated to use the scrambled identifiers, ensuring that any system interacting with WikiScramble encounters only the opaque schema. This scrambling is strictly harder than replacing IDs with random strings: the reassigned IDs such as `P9837` are real Wikidata identifiers that the model has prior associations with, so correctly resolving a predicate requires using the information provided through the tools—as instructed—rather than falling back on parametric knowledge of the original ID meanings.

In summary, WikiScramble consists of two artifacts: (1) a set of (blueprint, Wikidata-answer)-pairs, and (2) a scrambled version of Wikidata.

Evaluation. We ran BlueprintMiner on each blueprint in the benchmark to obtain a translated SPARQL query, then evaluated the SPARQL query on WikiData to obtain a set of WikiData entities. We compared these with the ground truth set of entities in the benchmark. While the `owl:sameAs` links are only partial, in particular Wikidata contains substantially more entities than DBpedia, so many Wikidata entities have no DBpedia counterpart, when comparison shows substantial overlap in their mapped entities, this provides a strong signal of semantic equivalence: the queries retrieve the same real-world objects despite using different schemas. A translation is considered correct at threshold τ if at least τ% of the ground truth WikiData entities appear in the predicted set. We report results across multiple thresholds to account for discrepancies due to differences in knowledge graph coverage.

5.7.2 Goby Benchmark Setup

Goby [63] is an enterprise relational dataset derived from a real-world production workload in the event promotion and marketing domain. Unlike Wikidata and DBpedia, which are public knowledge graphs that LLMs may have overfit to, Goby represents a genuinely private schema: its table and column names are unfamiliar to any pretrained model. It comprises 18,762 columns and 5.8M rows across multiple tables (Table 5.1). To preserve this property, the authors distribute the dataset encrypted at rest, preventing it from appearing as plaintext in large-scale web crawls such as The Pile [39] and thus from entering future LLM training corpora. Goby comes with no query blueprints. To evaluate BlueprintMiner on it, we manually authored a set of 25 query blueprints together with their ground truth answer sets, covering a range of join patterns and filter conditions representative of enterprise analytical queries. As described in Section 5.6, Query Blueprints extend naturally to relational data: each table is treated as a class, each row as an entity, and each column as a predicate, allowing the same translation pipeline to operate over SQL without modification.

5.7.3 WikiScramble Experiments

Setup. We evaluated BlueprintMiner on WikiScramble and compared it against four SPARQL agent baselines: GPT-4.1-mini, GPT-5-nano, Gemma4 (31B parameters), and Gemini-3.1-lite. Each agent is equipped with three tools: semantic search over predicates, semantic search over entities, and the ability to execute arbitrary SPARQL queries against WikiScramble, with results returned as JSON. All schema and instance data available to BlueprintMiner is equally available to the agents. Each agent is allowed up to 40 tool calls per query before timing out. Agent prompts include full tool documentation and explicit instructions to explore the data using the provided tools rather than assuming familiarity with any particular knowledge graph. This design is representative of

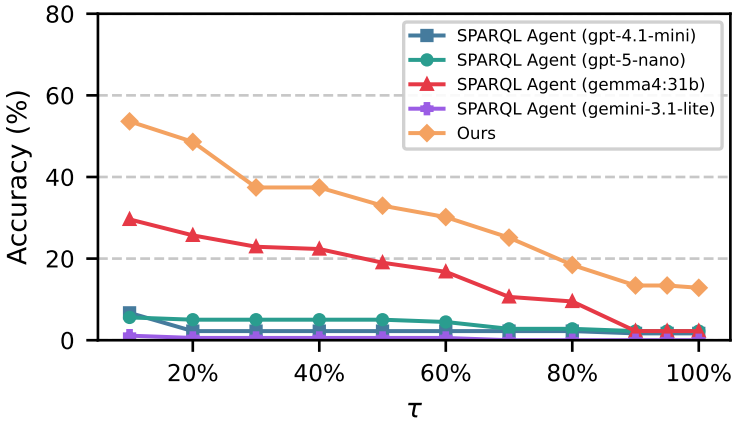


Figure 5.10: Evaluation on WikiScramble. At $\tau = 80\%$, all four agent baselines answer fewer than 10% of queries correctly (Gemma4: 9.4%, GPT-4.1-mini: 2.2%, GPT-5-nano: 2.8%; Gemini-3.1-lite answers 0.6% at $\tau = 60\%$). Query Blueprints answers 35% correctly at $\tau = 80\%$, rising to 48% at $\tau = 50\%$.

off-the-shelf SPARQL agent implementations [110]. All systems receive the same query blueprints and must produce executable SPARQL queries on WikiScramble.

Query Blueprints outperforms agentic approaches on WikiScramble.

Figure 5.10 presents our main results. Query Blueprints answers 35% of queries correctly at $\tau = 80\%$ and 48% at $\tau = 50\%$, while all four agent baselines answer fewer than 10% of queries at $\tau = 80\%$. Although standard Wikidata IDs such as P31 appear syntactically opaque, LLMs have overfit to public datasets; WikiScramble removes this shortcut, revealing that the agents' apparent schema reasoning relies on memorized IDs in their parametric knowledge.

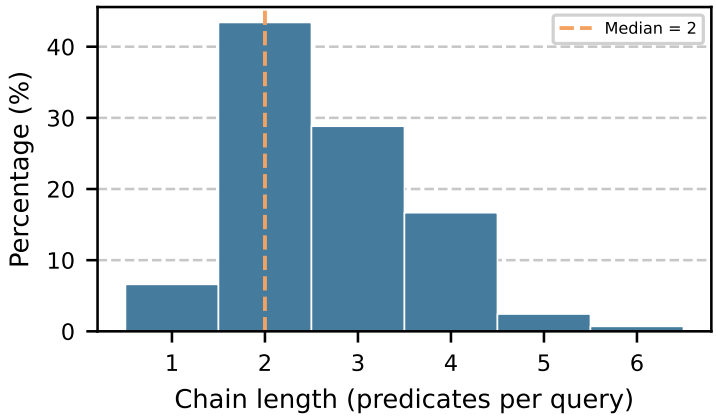


Figure 5.11: Distribution of total predicate chain lengths across translated queries. The average chain length per blueprint predicate is 1.78, reflecting frequent non-isomorphic translations in which a single blueprint predicate maps to a multi-hop path in Wikidata.

Translated queries are frequently non-isomorphic to their blueprints.

Figure 5.11 shows the distribution of total predicate chain lengths in the translated queries, ranging from 1 to 6. Translation increases the number of predicates by a factor of 1.78× on average: a single blueprint predicate frequently expands to a multi-hop path in Wikidata rather than mapping directly to one predicate. This non-isomorphic translation is a core challenge that motivates Query Blueprints: the system must discover structural correspondences that go beyond one-to-one predicate matching.

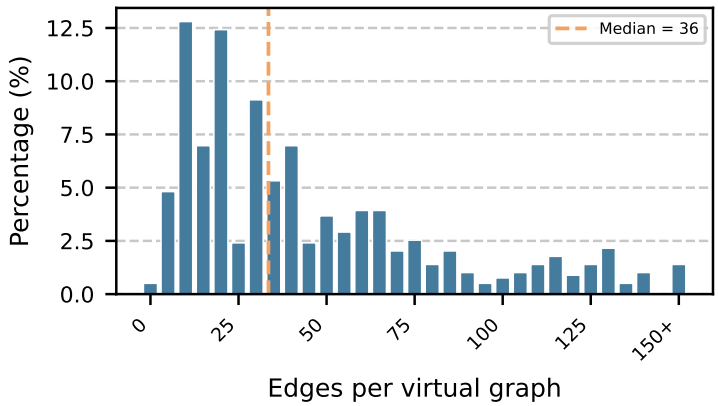


Figure 5.12: Distribution of virtual graph sizes (edges) across all queries. The median of 36 edges reflects a rich but tractable search space for pattern mining.

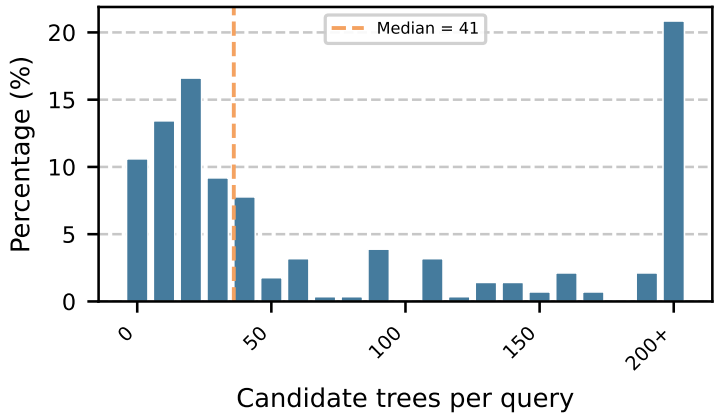


Figure 5.13: Distribution of candidate query trees considered per blueprint. The median is 41, with over 20% of blueprints generating 200 or more candidates, motivating the need for an effective selection strategy.

Virtual graphs are large and highly variable.

Figures 5.12 and 5.13 characterize the virtual graphs (Section 5.5.3) and the candidate trees mined from them. Virtual graphs have a median size of 36 edges, with a long tail extending past 150 (Figure 5.12). The median number of candidate trees per blueprint is 41, with over 20% of blueprints

generating 200 or more candidates (Figure 5.13), reflecting the combinatorial diversity of possible multi-hop translations.

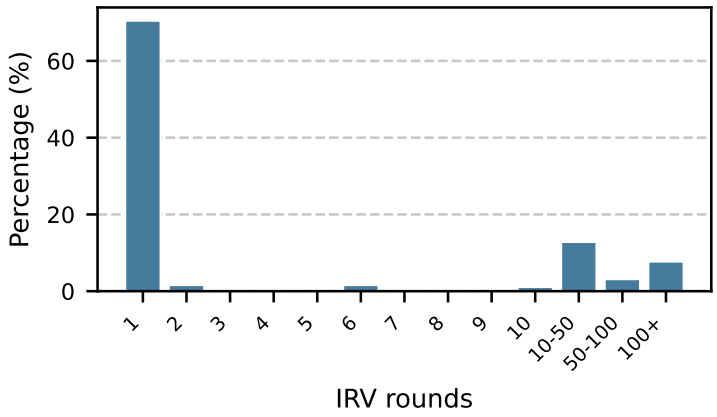


Figure 5.14: Distribution of IRV elimination rounds required to select a winning pattern. Most blueprints (70%) resolve in one round; 20% require 10 or more rounds, reflecting genuine ambiguity in the candidate space.

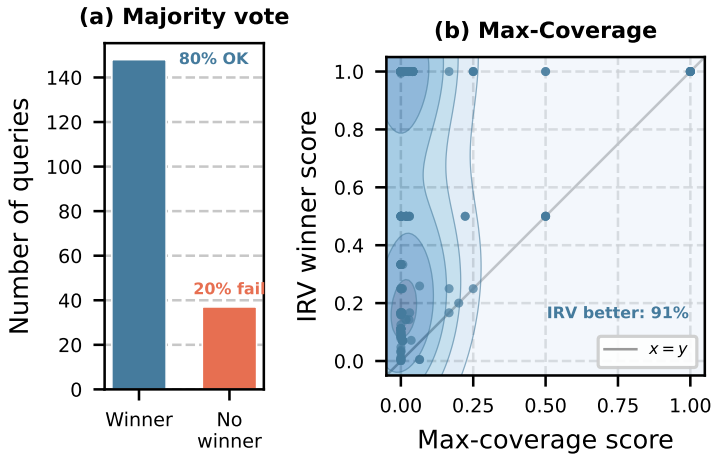


Figure 5.15: Justification for IRV voting over two baselines: majority vote and max-coverage winner. (a) Majority voting fails to produce a winner for 20% due to candidate diversity. (b) Max-coverage does produce a winner for every query, but IRV produces a higher-quality winner in 91% of cases.

IRV outperforms simpler voting strategies.

Figure 5.14 shows that 70% of blueprints are resolved in a single IRV round, reflecting clear agreement among examples. However, 20% require 10 or more rounds and 5% require over 100, reflecting genuine ambiguity in the candidate space. Figure 5.15 justifies this choice of voting strategy against two simpler alternatives. Majority voting—requiring a single candidate to receive more votes than all others combined—fails to produce a winner for 20% of blueprints due to candidate diversity (Figure 5.15a). Among the remaining blueprints, selecting the max-coverage candidate (the

pattern covering the most examples) yields a worse score than IRV in 91% of cases (Figure 5.15b), demonstrating that coverage alone is an insufficient criterion and that IRV's iterative elimination is essential for finding high-quality patterns.

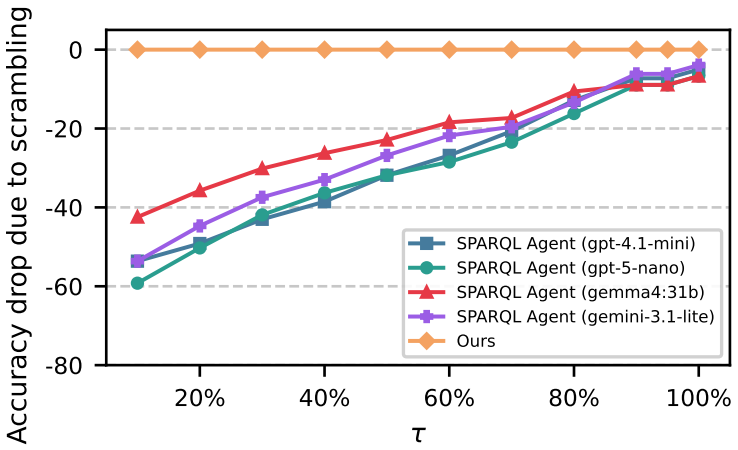


Figure 5.16: Ablation: unscrambling WikiScramble (restoring real Wikidata predicate IDs). All four agent baselines recover substantially; Query Blueprints remain unaffected, confirming that our approach does not overfit to public dataset schemas.

Table 5.2: Agent baseline statistics (WikiScramble).

	Gemma4	**GPT-4.1 -mini**	**GPT-5 -nano**	**Gemini-3.1-lite**	**Ours**
Tool calls per query					
Average	25.2	24.0	7.1	39.7	—
Median	23	24.5	7	40	—
Min / Max	4 / 40	3 / 40	1 / 19	8 / 40	—
Total tokens	35.6M	11.8M	7.61M	50.6M	**207K**
Total cost	$5.40	$5.01	$0.83	$13.03	**$0.40**

Agent collapse is explained by overfitting to public schema IDs.

Figure 5.16 shows an ablation in which we *unscramble* WikiScramble by restoring real Wikidata predicate IDs. All four agent baselines recover substantially, while Query Blueprints performance remains unchanged. This confirms that the agents' collapse on WikiScramble is driven by overfitting to public schema identifiers. The failure mode is distinctive: unable to find the IDs stored in their parametric knowledge, the agents repeatedly issue the same tool calls, searching for predicates like "instance of" that no longer exist under their expected identifiers, with average tool calls per query roughly doubling for GPT-4.1-mini and Gemma4 and frequently hitting the maximum limit of 40 (Table 5.2). This repetitive behavior is also reflected in token consumption: scrambling increases total tokens by 2.1× for GPT-5-nano (3.7M to 7.6M), 2.9× for GPT-4.1-mini (4.0M to 11.8M), 9.6× for Gemma4 (3.7M to 35.6M), and 1.5× for Gemini-3.1-lite (33.9M to 50.6M). The agents are unable to overcome the Wikidata prior, issuing wrong queries despite explicit prompts to not assume a particular knowledge graph and to trust the tools provided: `Never guess or hallucinate property IDs`

`or entity IDs...You must use tools to obtain the information you need.` Due to this, 3 of the 4 agents frequently hit the tool call limit of 40.

5.7.4 Goby Experiments

Setup. We loaded the Goby data into PostgreSQL and exposed it via a SPARQL interface using a relational-to-RDF view, as described in Section 5.7.2. We compare against a Gemini-based agent running on AlloyDB on Google Cloud (a PostgreSQL-compatible managed database), designed specifically for querying enterprise relational data.

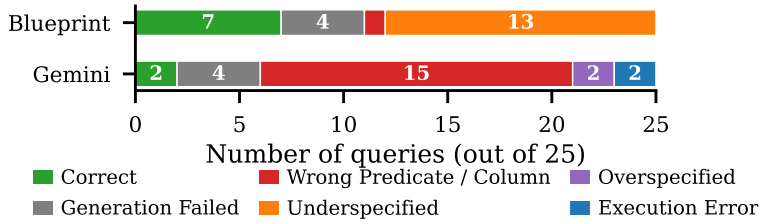


Figure 5.17: Error breakdown on the Goby benchmark (25 queries). Query Blueprints answers 7 queries correctly (28%) versus 2 for Gemini (8%). Gemini's dominant failure mode is wrong predicate/column (15 queries, 60%); Query Blueprints' dominant failure mode is partially incorrect results due to underspecification (13 queries, 52%).

Query Blueprints outperforms the Gemini agent on Goby.

Figure 5.17 shows the error breakdown across the 25 benchmark queries. Query Blueprints answers 7 queries correctly (28%), compared to 2 for Gemini (8%). The difference reflects a fundamental contrast in failure modes: Gemini can express complex SQL—multi-table UNIONs, aggregations, LIKE-based filters, multi-joins—but overwhelmingly points these structures at the wrong columns (15 queries, 60%), demonstrating that structural expressiveness without schema grounding produces little value.

Blueprint results are partially incorrect due to underspecification; Gemini's are wrong.

Query Blueprints produces partially incorrect results for 13 queries (52%), where the translated query is too broad and returns correct answers alongside spurious ones. A further 4 queries (16%) fail at generation because insufficient pattern examples were found, and only 1 query (4%) produces an outright wrong predicate. Crucially, Query Blueprints never overspecifies—this is precluded by the architecture, since an overspecified pattern would be rejected by the correlated examples. Gemini's failures are more severe: 15 queries (60%) return results against entirely wrong columns, 2 (8%) overspecify, 2 (8%) time out or error, and 4 (16%) produce no output at all.

Limitations The WikiScramble metric is recall-only: a translation counts as correct at threshold τ when at least τ% of the ground-truth entities appear in the returned set, with no penalty for entities returned in addition. Measuring precision would be the wrong instrument here, because Wikidata is broadly a superset of DBpedia—a faithful translation legitimately returns entities the source query could not, so an extra answer is more often evidence of the target's better coverage than of a bad translation. What the metric cannot do is separate that case from the one where the mined pattern was simply too general. The Goby breakdown of Figure 5.17, which counts partially correct answers as incorrect, is the stricter of the two readings.

Scrambling identifiers is also a weaker intervention than it may appear. WikiScramble is still Wikidata underneath: renaming the predicates removes the model's ability to retrieve one by name, but the graph's structure is untouched—which predicates co-occur, how a company is connected to its industry, what a plausible neighbourhood around an entity looks like—and that structure is in the training corpus regardless of what the identifiers are called. The benchmark therefore isolates vocabulary memorization specifically, not familiarity with the schema in general. A graph the model had never encountered in any form would be a harder test than the one reported here.

Finally, BlueprintMiner translates a blueprint into a single query pattern. Some questions cannot be answered by one: the same relationship is recorded through different predicate chains for different entities, and the full answer requires the union of several patterns. Because the voting stage of Section 5.5.5 elects one winner, no union can be expressed, and on such queries the system returns a proper subset of the intended answers whichever pattern wins. Supporting disjunctive translations is the clearest extension of the method.

5.8 Summary

Query blueprints factor natural-language query translation into two parts and assign each to the party that can supply it: the user states the structure exactly, in formal syntax, and BlueprintMiner grounds the vocabulary in the data by mining patterns from self-generated examples. On WikiScramble, where identifier scrambling removes the schema vocabulary from the model's parametric knowledge, this reaches 34% accuracy against under 10% for four state-of-the-art end-to-end agents, at $0.40 against $13.03 for the benchmark. On the relational Goby data of Chapter 4, it answers 28% of queries against the agent's 8%.

The factoring is not free, and the cost falls on the user. A blueprint must be written, which presumes an analyst fluent enough in SPARQL or SQL to express a join and a filter—a lower bar than knowing 57,896 predicate identifiers, but a higher one than typing an English sentence. Whether that trade is worth making depends on who is asking the question, and the design deliberately targets the analyst rather than the casual user. That cost, though, is one an agent may be able to absorb. Authoring a blueprint demands exactly the capability a language model has—turning an English question into a query shape—and none of the capability it lacks, because a blueprint names no schema identifiers. An agent placed in front of BlueprintMiner would restore the natural-language interface without reopening the failure mode of Section 5.2: it cannot hallucinate a predicate it is never asked to produce, and the vocabulary remains the system's job to mine. The architecture already admits this, accepting blueprints from a user or an agent alike (Figure 5.3). Whether an agent writes them as faithfully as an analyst is untested here, and is the most direct route to widening the method's audience.

The 34% figure deserves to be read carefully. It is an order of magnitude above the agents, and it is achieved on a schema constructed to be maximally hostile, but it is not a deployable accuracy. The remaining errors are informative about where the method is weak: on the relational benchmark, 52% of queries are *partially* correct, returning the intended answers together with spurious ones, because the mined pattern was too general. Underspecification, not misidentification, is the dominant failure—outright wrong predicates account for a single query. This is the opposite profile from the LLM baseline, whose queries are structurally fluent and pointed at entirely wrong columns 60% of the time, and it is a consequence of the architecture: an overspecified pattern is rejected by the correlated examples, while an underspecified one survives them. Tightening that boundary—by mining with more examples, or by scoring patterns on their selectivity as well as their support—is the most direct route to closing the gap.

Chapter 6

Conclusion

This dissertation set out to determine how foundation models change the treatment of semantic heterogeneity: which data-management tasks they help solve, and how they compare against the schema-matching and data-integration methods that preceded them. The three systems developed here—Chorus (Chapter 3), Goby (Chapter 4), and *Query Blueprints* (Chapter 5)—together return a qualified answer. On public schemas, foundation models subsume a generation of task-specific deep models. On enterprise schemas, the picture inverts: the same models fail, and recovering their performance requires grounding them in data rather than relying on what they have memorized.

6.1 Synthesis

The three contributions form a progression from the public setting to the private one.

Chorus establishes the ceiling that generalization alone can reach. A single prompt-based architecture, with no per-task training, matches or exceeds Sherlock, DoDuo, and TaBERT on table-class detection, column-type annotation, and join-column prediction, reaching 0.93, 0.89, and 0.90 F_1 on the three tasks. The lesson is that, when the schema vocabulary is present in the training corpus, a foundation model's general knowledge is a substitute for—not merely a supplement to—tens of thousands of task-specific labels.

Goby establishes that this ceiling is partly an artifact of the benchmark. Public benchmarks over-represent the public-web domains that foundation models have already memorized. On data drawn from a real enterprise workload, the same LLM- and ML-based methods regress by roughly 0.1 to 0.2 F_1 points. The regression is not fatal: treating the ontology as something to be *learned* from data samples, and presenting its full hierarchy to the model through tree serialization, recovers the lost performance. The lesson here is as much methodological as empirical—the field's optimism about LLMs for data integration has been calibrated against benchmarks that do not resemble the data on which these systems are deployed.

Query Blueprints establishes what to do when memorization is unavailable by construction. By factoring a query into its *structure*, which the user states exactly, and its *vocabulary*, which the system grounds in the data rather than in the model's parameters, it answers queries over opaque schemas where end-to-end agents collapse: 34% accuracy against under 10% for four state-of-the-art LLM agents, at $0.40 against $13.03 in cost. The lesson is architectural: on an unfamiliar schema, the most reliable use of a language model is the most limited one.

The arc connecting the three was not imposed in retrospect. Goby quantified an enterprise gap and closed it for a single discovery task; its central finding—that grounding in data beats parametric knowledge—is precisely what *Query Blueprints* carries into the harder problem of querying. Each

system inherits the question the previous one left open.

6.2 Cross-Cutting Lessons

Three themes recur across the contributions and outlast the specific systems that exhibit them.

Grounding beats parametric knowledge. The single finding that organizes this dissertation is that a foundation model's value on a heterogeneous schema depends entirely on whether that schema is in its training data. When the schema is public, parametric knowledge suffices, and CHORUS wins. When it is private, parametric knowledge fails silently—an invented predicate or a hallucinated join produces a confident wrong answer—and only grounding works. Hallucination is not an incidental defect to be engineered away; under mild assumptions it is an unavoidable property of calibrated models [58]. The productive response is not to suppress it but to remove the model from the loop wherever the data can answer the question instead.

Know what to unify and what to factor. CHORUS improves accuracy by *unifying* three discovery tasks in one context, so that the output of table-class detection disambiguates the column annotations that follow. *Query Blueprints* improves accuracy by *factoring* a single task—query translation—into structure and vocabulary, so that the user's intent is never corrupted by the system's schema guesses. The two moves are opposite in form but identical in spirit: each isolates the part of the problem a foundation model handles well from the part it handles badly, and routes each part to the mechanism suited to it.

Use the language model sparingly. The most robust design point found in this work is a neuro-symbolic division of labor: the LLM supplies open-ended generalization and example generation, and classical algorithms supply grounding and verification. *Query Blueprints* confines the model to seed-example generation and recovers schema matching, keyword search, and social-choice voting for everything else; CHORUS wraps the model in constraint checks and the *anchoring* mitigation rather than trusting its raw output. In both, the decades of database technique surveyed in Chapter 2 are not made obsolete by foundation models but reorganized around them.

6.3 Open Directions

Scale. All three systems were evaluated on schemas smaller than the 10^5-attribute deployments that motivate the problem. Whether the techniques hold at that scale, and at the data volumes of an enterprise lake, is open. Tree serialization in particular is context-hungry: presenting a full ontology hierarchy that grows past the context window will require principled summarization rather than the ad-hoc truncation used today. Cost and throughput, manageable on the benchmarks here, are magnified by the extensive duplication characteristic of enterprise data.

An end-to-end pipeline. The three systems address consecutive stages—discovering a schema, learning its ontology, and querying it—but were built and evaluated in isolation. A system that runs CHORUS to annotate an unfamiliar schema, GOBY's procedure to synthesize an ontology over it, and *Query Blueprints* to answer queries against that ontology would test whether the contributions compose, and whether errors compound or cancel across the stages.

Beyond annotation and single queries. The tasks studied here are discovery and translation. Richer integration tasks—data exchange, provenance, and update propagation—demand reasoning that composes many more steps than column annotation or path discovery, and CHORUS's pattern-matching hypothesis predicts that foundation models will resist them. Identifying which data-management problems lie inside and outside the reach of grounded language-model methods is itself a research program.

Realistic, contamination-resistant benchmarks. GOBY is one benchmark, from one industry, releasable only because it could be scraped and relabelled. The community needs more semi-private benchmarks, and protocols—such as temporal hold-out and identifier scrambling, both used in this dissertation—that keep evaluation honest as today's test sets leak into tomorrow's training corpora. Without them, the enterprise gap will continue to be discovered anew, one deployment at a time [37].

The throughline of this dissertation is a single reversal. Foundation models arrived promising to make data management easier by knowing more than any schema could teach them. On the data that matters most—private, opaque, and absent from every training corpus—they know almost nothing, and their fluency becomes a liability rather than an asset. What recovers them is old: grounding claims in the data, factoring problems into parts, and trusting the model only where it has earned trust. Semantic heterogeneity is not dissolved by scale. It is managed, as it always has been, by understanding the data—and the new tools work best when they are bent to that end.

Bibliography

[1] Nora Abdelmageed, Jiaoyan Chen, Vincenzo Cutrona, Vasilis Efthymiou, Oktie Hassanzadeh, Madelon Hulsebos, Ernesto Jiménez-Ruiz, Juan Sequeda, and Kavitha Srinivas. 2022. Results of SemTab 2022. In *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching, SemTab 2021, co-located with the 21st International Semantic Web Conference, ISWC 2022, Virtual conference, October 23-27, 2022 (CEUR Workshop Proceedings, Vol. 3320)*, Vasilis Efthymiou, Ernesto Jiménez-Ruiz, Jiaoyan Chen, Vincenzo Cutrona, Oktie Hassanzadeh, Juan Sequeda, Kavitha Srinivas, Nora Abdelmageed, and Madelon Hulsebos (Eds.). CEUR-WS.org, 1–13. `https://ceur-ws.org/Vol-3320/paper0.pdf`

[2] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. 2002. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, Rakesh Agrawal and Klaus R. Dittrich (Eds.). IEEE Computer Society, 5–16. doi:`10.1109/ICDE.2002.994693`

[3] Inc. Anaconda. 2021. State of Data Science. https://www.anaconda.com/resources/whitepapers/state-of-data-science-2021.

[4] Rexer Analytics. 2023. Data Science Survey. `https://www.rexeranalytics.com/data-science-survey`

[5] Jacob Andreas. 2022. Language Models as Agent Models. arXiv:2212.01681 [cs.CL]

[6] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. arXiv:2304.09433 [cs.CL]

[7] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proc. VLDB Endow.* 17, 2 (2023), 92–105. `https://www.vldb.org/pvldb/vol17/p92-arora.pdf`

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The Semantic Web. *Scientific American* 284, 5 (2001), 34–43.

[9] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. 2011. Generic Schema Matching, Ten Years Later. *Proc. VLDB Endow.* 4, 11 (2011), 695–701. `http://www.vldb.org/pvldb/vol4/p695-bernstein_madhavan_rahm.pdf`

[10] Philip A. Bernstein, Sergey Melnik, Michalis Petropoulos, and Christoph Quix. 2004. Industrial-Strength Schema Matching. *SIGMOD Rec.* 33, 4 (2004), 38–43. doi:`10.1145/1041410.1041417`

[11] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ B. Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. 2021. On the Opportunities and Risks of Foundation Models. *CoRR* abs/2108.07258 (2021). arXiv:2108.07258 `https://arxiv.org/abs/2108.07258`

[12] Dan Brickley, Matthew Burgess, and Natasha F. Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia (Eds.). ACM, 1365–1375. doi:`10.1145/3308558.3313685`

[13] Roland Bullivant. 2018. *GDPR is Coming – Do You Need to Find the Location of Personal Data in Your ERP and CRM Systems?* Technical Report. Silwood Technology.

[14] Michael J. Cafarella, Alon Y. Halevy, Hongrae Lee, Jayant Madhavan, Cong Yu, Daisy Zhe Wang, and Eugene Wu. 2018. Ten Years of WebTables. *Proc. VLDB Endow.* 11, 12 (2018), 2140–2149. doi:`10.14778/3229863.3240492`

[15] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: exploring the power of tables on the web. *Proc. VLDB Endow.* 1, 1 (2008), 538–549. doi:`10.14778/1453856.1453916`

[16] Sonia Castelo, Rémi Rampin, Aécio S. R. Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. 2021. Auctus: A Dataset Search Engine for Data Discovery and Augmentation. *Proc. VLDB Endow.* 14, 12 (2021), 2791–2794. doi:`10.14778/3476311.3476346`

[17] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. 2020. Dataset search: a survey. *VLDB J.* 29, 1 (2020), 251–272. doi:`10.1007/s00778-019-00564-x`

[18] Zhimin Chen, Vivek R. Narasayya, and Surajit Chaudhuri. 2014. Fast Foreign-Key Detection in Microsoft SQL Server PowerPivot for Excel. *Proc. VLDB Endow.* 7, 13 (2014), 1417–1428. doi:`10.14778/2733004.2733014`

[19] Gari D. Clifford, Daniel J. Scott, and Mauricio Villarroel. 2009. *Documentation for the MIMIC II Database*. Technical Report. Harvard-MIT Division of Health Sciences & Technology. `https://archive.physionet.org/mimic2/UserGuide/UserGuide.pdf` 76 pages.

[20] Jean-Antoine-Nicolas de Caritat Condorcet. 1785. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Imprimerie Royale, Paris.

[21] Jean-Antoine-Nicolas de Caritat Condorcet. 1788. On the Constitution and the Functions of Provincial Assemblies. In *Œuvres complètes de Condorcet*. Vol. 13. 243. Published 1804.

[22] Tianji Cong, James Gale, Jason Frantz, H. V. Jagadish, and Çagatay Demiralp. 2022. WarpGate: A Semantic Join Discovery System for Cloud Data Warehouses. *CoRR* abs/2212.14155 (2022). arXiv:2212.14155 doi:10.48550/arXiv.2212.14155

[23] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. 2002. Mining database structure; or, how to build a data quality browser. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, June 3-6, 2002*, Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki (Eds.). ACM, 240–251. doi:10.1145/564691.564719

[24] Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. 2024. Investigating Data Contamination in Modern Benchmarks for Large Language Models. In *NAACL*. 8706–8719.

[25] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (2020), 307–319. doi:10.5555/3430915.3442430

[26] Amol Deshpande. 2025. Beyond Relations: A Case for Elevating to the Entity-Relationship Abstraction. *CIDR* (2025).

[27] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG]

[28] AnHai Doan and Alon Y. Halevy. 2005. Semantic Integration Research in the Database Community: A Brief Survey. *AI Mag.* 26, 1 (2005), 83–94. doi:10.1609/AIMAG.V26I1.1801

[29] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration*. Morgan Kaufmann. http://research.cs.wisc.edu/dibook/

[30] Jesse Dodge, Maarten Sap, Ana Marasovic, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting Large Webtext Corpora: A Case Study on the Colossal Clean Crawled Corpus. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 1286–1305. doi:10.18653/v1/2021.emnlp-main.98

[31] Xin Luna Dong and Divesh Srivastava. 2015. *Big Data Integration*. Morgan & Claypool Publishers. doi:10.2200/S00578ED1V01Y201404DTM040

[32] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2022. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. *CoRR* abs/2212.07588 (2022). arXiv:2212.07588 doi:10.48550/arXiv.2212.07588

[33] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, and Yejin Choi. 2023. Faith and Fate: Limits of Transformers on Compositionality. *CoRR* abs/2305.18654 (2023). arXiv:2305.18654 doi:10.48550/arXiv.2305.18654

[34] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *CoRR* abs/2404.16130 (2024). arXiv:2404.16130 doi:10.48550/ARXIV.2404.16130

[35] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2003. Data Exchange: Semantics and Query Answering. In *Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8-10, 2003, Proceedings (Lecture Notes in Computer Science)*, Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani (Eds.). Springer, 207–224. doi:10.1007/3-540-36285-1_14

[36] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. 2023. Table Discovery in Data Lakes: State-of-the-art and Future Directions. In *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, Sudipto Das, Ippokratis Pandis, K. Selçuk Candan, and Sihem Amer-Yahia (Eds.). ACM, 69–75. doi:10.1145/3555041.3589409

[37] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. NL2SQL is a solved problem... Not!. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p74-floratou.pdf

[38] Common Crawl Foundation. 2011. Common Crawl. https://commoncrawl.org

[39] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2021. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *CoRR* abs/2101.00027 (2021). arXiv:2101.00027 https://arxiv.org/abs/2101.00027

[40] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 3356–3369. doi:10.18653/v1/2020.findings-emnlp.301

[41] Allan Gibbard. 1973. Manipulation of Voting Schemes: A General Result. *Econometrica* 41, 4 (1973), 587–601. doi:10.2307/1914083

[42] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance Semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Beijing, China) *(PODS '07)*. Association for Computing Machinery, New York, NY, USA, 31–40.

[43] Ken Gu, Advait Bhat, Mike A. Merrill, Robert West, Xin Liu, Daniel McDuff, and Tim Althoff. 2025. SynthWorlds: Controlled Parallel Worlds for Disentangling Reasoning and Knowledge in Language Models. *CoRR* abs/2510.24427 (2025). arXiv:2510.24427 doi:10.48550/ARXIV.2510.24427

[44] Mike Gualtieri and Noel Yuhanna. 2016. *The Forrester Wave: Big Data Hadoop Distributions, Q1 2016*. Forrester Research, Inc.

[45] Arnav Gudibande, Eric Wallace, Charlie Snell, Xinyang Geng, Hao Liu, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. The False Promise of Imitating Proprietary LLMs. arXiv:2305.15717 [cs.CL]

[46] Steven Harris and Andy Seaborne. 2013. *SPARQL 1.1 Query Language*. W3C Recommendation. W3C. https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[47] P. Bryan Heidorn. 2008. Shedding light on the dark data in the long tail of science. *Library trends* 57, 2 (2008), 280–299.

[48] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. `https://openreview.net/forum?id=rygGQyrFvH`

[49] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. 2003. Efficient IR-Style Keyword Search over Relational Databases. In *Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, September 9-12, 2003*, Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer (Eds.). Morgan Kaufmann, 850–861. doi:`10.1016/B978-012722442-8/50080-X`

[50] Kevin Hu, Neil Gaikwad, Michiel Bakker, Madelon Hulsebos, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. 2019. VizNet: Towards a large-scale visualization learning and benchmarking repository. In *Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI)*. ACM.

[51] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. *arXiv preprint arXiv:2201.07207* (2022).

[52] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2021. GitTables: A Large-Scale Corpus of Relational Tables. *CoRR* abs/2106.07258 (2021). arXiv:2106.07258 `https://arxiv.org/abs/2106.07258`

[53] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. In *SIGMOD*.

[54] Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 1500–1508. doi:`10.1145/3292500.3330993`

[55] Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Özcan, Yannis Papakonstantinou, and Reynold Cheng. 2025. BIRD-INTERACT: Re-imagining Text-to-SQL Evaluation for Large Language Models via Lens of Dynamic Interactions. *CoRR* abs/2510.05318 (2025). arXiv:2510.05318 doi:`10.48550/ARXIV.2510.05318`

[56] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 3446–3456. doi:10.18653/v1/2021.naacl-main.270

[57] Nandish Jayaram, Arijit Khan, Chengkai Li, Xifeng Yan, and Ramez Elmasri. 2015. Querying Knowledge Graphs by Example Entity Tuples. *IEEE Trans. Knowl. Data Eng.* 27, 10 (2015), 2797–2811. doi:10.1109/TKDE.2015.2426696

[58] Adam Tauman Kalai and Santosh S. Vempala. 2024. Calibrated Language Models Must Hallucinate. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, Bojan Mohar, Igor Shinkar, and Ryan O'Donnell (Eds.). ACM, 160–171. doi:10.1145/3618260.3649777

[59] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare (Eds.). ACM, 3363–3372. doi:10.1145/1978942.1979444

[60] Nikhil Kandpal, Eric Wallace, and Colin Raffel. 2022. Deduplicating Training Data Mitigates Privacy Risks in Language Models. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 10697–10707. https://proceedings.mlr.press/v162/kandpal22a.html

[61] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2023. CHORUS: Foundation Models for Unified Data Discovery and Exploration. *CoRR* abs/2306.09610 (2023). arXiv:2306.09610 doi:10.48550/arXiv.2306.09610

[62] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2024. Chorus: Foundation Models for Unified Data Discovery and Exploration. *Proc. VLDB Endow.* 17, 8 (may 2024), 2104–2114. doi:10.14778/3659437.3659461

[63] Moe Kayali, Fabian Wenz, Nesime Tatbul, and Çagatay Demiralp. 2025. Mind the Data Gap: Bridging Large Language Models (LLMs) to Enterprise Data Integration. In *15th Conference on Innovative Data Systems Research, CIDR 2025, Amsterdam, The Netherlands, January 19-22, 2025*. www.cidrdb.org. https://vldb.org/cidrdb/2025/mind-the-data-gap-bridging-llms-to-enterprise-data-integration.html

[64] Arijit Khan. 2023. Knowledge Graphs Querying. *SIGMOD Rec.* 52, 2 (2023), 18–29. doi:10.1145/3615952.3615956

[65] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1 (2023), 9:1–9:25. doi:10.1145/3588689

[66] Aneta Koleva, Martin Ringsquandl, Mitchell Joblin, and Volker Tresp. 2021. Generating Table Vector Representations. *CoRR* abs/2110.15132 (2021). arXiv:2110.15132 https://arxiv.org/abs/2110.15132

[67] Nicholas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. 1997. Wrapper Induction for Information Extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*. Morgan Kaufmann, 729–737.

[68] Ellen J Langer, Arthur Blank, and Benzion Chanowitz. 1978. The mindlessness of ostensibly thoughtful action: The role of" placebic" information in interpersonal interaction. *Journal of personality and social psychology* 36, 6 (1978), 635.

[69] David Langford. 1999. COMP.BASILISK FAQ. *Nature* 402, 6761 (1999), 465–465. doi:10.1038/44964

[70] Maurizio Lenzerini. 2002. Data Integration: A Theoretical Perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis (Eds.). ACM, 233–246. doi:10.1145/543613.543644

[71] Vladimir Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In *Soviet Physics Doklady*, Vol. 10. 707.

[72] Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 3214–3252. doi:10.18653/v1/2022.acl-long.229

[73] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. 2001. Generic Schema Matching with Cupid. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass (Eds.). Morgan Kaufmann, 49–58. http://www.vldb.org/conf/2001/P049.pdf

[74] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. 2022. When Not to Trust Language Models: Investigating Effectiveness and Limitations of Parametric and Non-Parametric Memories. arXiv:2212.10511 [cs.CL]

[75] Pablo N. Mendes, Max Jakob, and Christian Bizer. 2012. DBpedia: A Multilingual Cross-domain Knowledge Base. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association (ELRA), 1813–1817. http://www.lrec-conf.org/proceedings/lrec2012/summaries/570.html

[76] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. 2014. Exemplar Queries: Give me an Example of What You Need. *Proc. VLDB Endow.* 7, 5 (2014), 365–376. doi:10.14778/2732269.2732273

[77] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.* 16, 4 (2022), 738–746. https://www.vldb.org/pvldb/vol16/p738-narayan.pdf

[78] Fatemeh Nargesian, Ken Q. Pu, Bahar Ghadiri Bashardoost, Erkang Zhu, and Renée J. Miller. 2023. Data Lake Organization. *IEEE Trans. Knowl. Data Eng.* 35, 1 (2023), 237–250. doi:10.1109/TKDE.2021.3091101

[79] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proc. VLDB Endow.* 12, 12 (2019), 1986–1989. doi:10.14778/3352063.3352116

[80] Washington State Department of Licensing. 2023. Electric Vehicle Population Data Electric Vehicle Population Data. https://catalog.data.gov/dataset/electric-vehicle-population-data

[81] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2022. In-context Learning and Induction Heads. *CoRR* abs/2209.11895 (2022). arXiv:2209.11895 doi:10.48550/ARXIV.2209.11895

[82] Reham Omar, Ishika Dhall, Panos Kalnis, and Essam Mansour. 2023. A Universal Question-Answering Platform for Knowledge Graphs. *Proc. ACM Manag. Data* 1, 1 (2023), 57:1–57:25. doi:10.1145/3588911

[83] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *CoRR* abs/2203.02155 (2022). arXiv:2203.02155 doi:10.48550/arXiv.2203.02155

[84] Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2025. Text2Cypher: Bridging Natural Language and Graph Databases. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025 - Workshops, Abu Dhabi, UAE, January 19-24, 2025*. Association for Computational Linguistics, 100–108. https://aclanthology.org/2025.genaik-1.11/

[85] Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. 2024. Grammar-Aligned Decoding. *CoRR* abs/2405.21047 (2024). arXiv:2405.21047 doi:10.48550/ARXIV.2405.21047

[86] Jeffrey Pound, Alexander K. Hudek, Ihab F. Ilyas, and Grant E. Weddell. 2012. Interpreting keyword queries over web knowledge bases. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, Xue-wen Chen, Guy Lebanon, Haixun Wang, and Mohammed J. Zaki (Eds.). ACM, 305–314. doi:10.1145/2396761.2396803

[87] Jeffrey Pound, Ihab F. Ilyas, and Grant E. Weddell. 2010. Expressive and flexible access to web-extracted data: a keyword-based structured query language. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, Ahmed K. Elmagarmid and Divyakant Agrawal (Eds.). ACM, 423–434. doi:10.1145/1807167.1807214

[88] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs] doi:10.48550/arXiv.2304.11015

[89] Neil Raden. 2013. *Success Stories with Teradata Analytics for SAP Solutions*. Technical Report. Hired Brains Research. https://assets.teradata.com/resourceCenter/downloads/AnalystReports/EB9439.pdf Report for Teradata.

[90] Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew J. Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. *CoRR* abs/2112.11446 (2021). arXiv:2112.11446 https://arxiv.org/abs/2112.11446

[91] Erhard Rahm and Philip A. Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDB J.* 10, 4 (2001), 334–350. doi:10.1007/S007780100057

[92] Erhard Rahm, Hong Hai Do, and Sabine Massmann. 2004. Matching Large XML Schemas. *SIGMOD Rec.* 33, 4 (2004), 26–31. doi:10.1145/1041410.1041415

[93] Dominique Ritze and Christian Bizer. 2017. Matching Web Tables To DBpedia - A Feature Utility Study. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, Volker Markl, Salvatore Orlando, Bernhard Mitschang, Periklis Andritsos, Kai-Uwe Sattler, and Sebastian Breß (Eds.). OpenProceedings.org, 210–221. doi:10.5441/002/edbt.2017.20

[94] Aécio S. R. Santos, Aline Bessa, Christopher Musco, and Juliana Freire. 2022. A Sketch-based Index for Correlated Dataset Search. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2928–2941. doi:10.1109/ICDE53745.2022.00264

[95] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. 2023. Large Language Models Can Be Easily Distracted by Irrelevant Context. *CoRR* abs/2302.00093 (2023). arXiv:2302.00093 doi:10.48550/arXiv.2302.00093

[96] Karan Singhal, Shekoofeh Azizi, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Kumar Tanwani, Heather Cole-Lewis, Stephen Pfohl, Perry Payne, Martin Seneviratne, Paul Gamble, Chris Kelly, Nathaneal Schärli, Aakanksha Chowdhery, Philip Andrew Mansfield, Blaise Agüera y Arcas, Dale R. Webster, Gregory S. Corrado, Yossi

Matias, Katherine Chou, Juraj Gottweis, Nenad Tomasev, Yun Liu, Alvin Rajkomar, Joelle K. Barral, Christopher Semturs, Alan Karthikesalingam, and Vivek Natarajan. 2022. Large Language Models Encode Clinical Knowledge. *CoRR* abs/2212.13138 (2022). arXiv:2212.13138 doi:10.48550/arXiv.2212.13138

[97] Michael Stonebraker, Daniel Bruckner, Ihab F. Ilyas, George Beskales, Mitch Cherniack, Stanley B. Zdonik, Alexander Pagan, and Shan Xu. 2013. Data Curation at Scale: The Data Tamer System. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2013/Papers/CIDR13_Paper28.pdf

[98] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating Columns with Pre-trained Language Models. In *Proceedings of the 2022 International Conference on Management of Data*. Association for Computing Machinery. https://doi.org/10.1145/3514221.3517906

[99] Daniel Sun and Rita Sallam. 2024. *Calculating the ROI on GenAI Business Model Innovation*. Technical Report. Gartner Research, https://www.gartner.com/en/documents/5517195.

[100] Kai Sun, Yifan Ethan Xu, Hanwen Zha, Yue Liu, and Xin Luna Dong. 2024. Head-to-Tail: How Knowledgeable are Large Language Models (LLMs)? A.K.A. Will LLMs Replace Knowledge Graphs?. In *NAACL*. 311–325.

[101] Ruoxi Sun, Sercan Ö Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. 2024. SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (Extended). arXiv:2306.00739 [cs] doi:10.48550/arXiv.2306.00739

[102] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A Large Language Model for Science. *CoRR* abs/2211.09085 (2022). arXiv:2211.09085 doi:10.48550/arXiv.2211.09085

[103] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023). arXiv:2302.13971 doi:10.48550/arXiv.2302.13971

[104] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open

Foundation and Fine-Tuned Chat Models. *CoRR* abs/2307.09288 (2023). arXiv:2307.09288 doi:10.48550/arXiv.2307.09288

[105] Trifacta. 2023. Trifacta Wrangler. https://cloud.trifacta.com. Accessed: 2023-04-10.

[106] Roy W. Tromble and Jason Eisner. 2006. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 4-9, 2006, New York, New York, USA*. https://aclanthology.org/N06-1054/

[107] Immanuel Trummer. 2021. Can Deep Neural Networks Predict Data Correlations from Column Names? *CoRR* abs/2107.04553 (2021). arXiv:2107.04553 https://arxiv.org/abs/2107.04553

[108] Immanuel Trummer. 2022. Towards NLP-Enhanced Data Profiling Tools. In *12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, January 9-12, 2022*. www.cidrdb.org. https://www.cidrdb.org/cidr2022/papers/a55-trummer.pdf

[109] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2023. *ChatGPT for Robotics: Design Principles and Model Abilities*. Technical Report MSR-TR-2023-8. Microsoft. https://www.microsoft.com/en-us/research/publication/chatgpt-for-robotics-design-principles-and-model-abilities/

[110] Web & Software Engineering Research Group, Leipzig University of Applied Sciences. 2025. Text2SPARQL Agent. https://github.com/WSE-research/text2sparql-agent.

[111] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. arXiv:2206.07682 [cs.CL]

[112] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *Trans. Mach. Learn. Res.* 2022 (2022). https://openreview.net/forum?id=yzkSU5zdwD

[113] Peter West, Chandra Bhagavatula, Jack Hessel, Jena D. Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. 2022. Symbolic Knowledge Distillation: from General Language Models to Commonsense Models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, Marine Carpuat, Marie-Catherine de Marneffe, and Iván Vladimir Meza Ruíz (Eds.). Association for Computational Linguistics, 4602–4625. doi:10.18653/v1/2022.naacl-main.341

[114] Niklas Wretblad, Fredrik Gordh Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. 2024. Understanding the Effects of Noise in Text-to-SQL: An Examination of the BIRD-Bench Benchmark. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, ACL 2024 - Short Papers, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 356–369. doi:10.18653/V1/2024.ACL-SHORT.34

[115] Cong Yan and Yeye He. 2020. Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1539–1554. doi:10.1145/3318464.3389738

[116] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 8413–8426. doi:10.18653/v1/2020.acl-main.745

[117] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 3911–3921. doi:10.18653/V1/D18-1425

[118] Ce Zhang, Jaeho Shin, Christopher Ré, Michael J. Cafarella, and Feng Niu. 2016. Extracting Databases from Dark Data with DeepDive. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 847–859. doi:10.1145/2882903.2904442

[119] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. *Proc. VLDB Endow.* 13, 11 (2020), 1835–1848. http://www.vldb.org/pvldb/vol13/p1835-zhang.pdf

[120] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. In *VLDB*.

[121] Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. 2023. How Language Model Hallucinations Can Snowball. arXiv:2305.13534 [cs.CL]

[122] Fuheng Zhao, Divyakant Agrawal, and Amr El Abbadi. 2024. Hybrid Querying Over Relational Databases and Large Language Models. *CoRR* abs/2408.00884 (2024). arXiv:2408.00884 doi:10.48550/ARXIV.2408.00884

[123] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate Before Use: Improving Few-shot Performance of Language Models. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 12697–12706. http://proceedings.mlr.press/v139/zhao21c.html

[124] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *CoRR* abs/2306.05685 (2023). arXiv:2306.05685 doi:10.48550/arXiv.2306.05685

[125] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 847–864. doi:10.1145/3299869.3300065