Journal Title
XX(X):1–23
©The Author(s) 0000
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

When does the user's wording break knowledge graph question answering? Vocabulary mismatch and underspecification in tool-based systems

Jakob Deich[1], Tobias Meisen[1] and André Pomp[1]

Abstract

Users of natural-language query interfaces do not know the schema vocabulary: they ask for *students* where it says `Learner`, misspell names, and leave details unstated. Text2SQL loses 20 to 30 points under synonym substitution, but for knowledge graph question answering (KGQA) no benchmark controls how far a question's wording departs from the schema vocabulary, and whether the loss persists for agentic systems, which search the schema before answering, is open. We study this for enterprise KGQA over R2RML/RML mappings with proprietary vocabularies. A two-agent system whose retrieval interface is swapped between lexical and embedding-based matching is measured under four input conditions, five datasets, and four LLM backbones. Tool access alone does not repair vocabulary mismatch: agents search 47 to 78% more and fail two to three times as often, yet still lose 22 to 30 points of F_1. Typographical errors leave result quality unchanged. Semantic retrieval recovers 0.09 to 0.12 data F_1 for three backbones; the strongest requests clarification instead of guessing. In all 240 uninstructed runs, underspecified questions were answered under a silently chosen reading; a one-sentence instruction recovers this for the two strongest backbones. We conclude with the properties a benchmark needs to make these effects measurable.

Keywords

enterprise knowledge graph question answering, Text2SPARQL, vocabulary mismatch, underspecification, LLM agents, benchmark design, ontology-based data access

1 Introduction

Text2Query systems translate a natural-language (NL) query into a formal query, either SQL over a relational database or SPARQL over a knowledge graph. However, such a system is useful only if it tolerates the way people really phrase questions. Often, users do not know the schema behind the interface. They ask for *students* where the vocabulary says `Learner` (we set user wording in italics and schema labels in monospace throughout), they misspell names, and they omit details that become necessary once the question is grounded in the data vocabulary. Every system must therefore bridge two gaps: one between the words of a question and the labels of a schema, and one between what the question states and what answering it requires. The SQL literature calls the first step *schema linking*[1;2]. The graph literature splits it into *entity linking* and *relation linking*[3;4].

Both fields have evidence that the step is fragile, but of different kinds. For Text2SQL the evidence is experimental: Gan et al.[5] replaced schema-related words in Spider questions with ordinary synonyms, left the target SQL untouched, and accuracy fell by 20 to 30 percentage points. For KGQA it is descriptive: Steinmetz and Sattler[6] surveyed 26 datasets and found that a quarter to a half of the entities, and more than half of the properties, a gold SPARQL query needs cannot be recovered from the question text at all (cf. Section 2.1). The survey shows how far questions and labels lie apart in existing datasets, but it cannot show how much result quality is lost at that distance, because no dataset offers a matched set of questions in which the distance is absent. Questions near the schema vocabulary and questions far from it are pooled into one test set and reported as one score, and that score cannot reveal how much of a result depends on questions that use the schema's vocabulary.

The Text2SQL finding also cannot simply be carried over to today's LLM-based systems, which no longer answer in a single pass. The models Gan et al. evaluated emit a query in one forward pass, so an unresolved term is lost with no recourse. Recent systems are built as agents, in Text2SQL[7–9] as well as in KGQA[10–12]. They call search tools over the schema, read what comes back, and revise before committing. An agent that fails to find "student" can try "learner", follow the class hierarchy, or search for something else. Whether this suffices to absorb a vocabulary

[1]Institute of Technologies and Management of Digital Transformation, University of Wuppertal, Germany

Corresponding author:
Jakob Deich, Institute of Technologies and Management of Digital Transformation, University of Wuppertal, Gaußstraße 20, 42119 Wuppertal, Germany.
Email: deich@uni-wuppertal.de

mismatch cannot be inferred from single-pass models, which neither search nor revise.

The problem is most consequential in settings that Text2Query benchmarks rarely cover. Benchmarks such as Spider[13], QALD[14], and LC-QuAD[15] target public schemas and graphs whose labels come from ordinary language, so question and label are seldom far apart to begin with. Spider 2.0[16] is a deliberate exception: it moves to real enterprise workflows with schemas of over a thousand columns and agents that consult metadata and documentation, and accuracy drops from over 90% to 21% as a result. Each of its tasks is nonetheless posed against a single database, so a query is grounded in the table and column names of one provider. Sequeda et al.[17] describe a related setting as question answering over enterprise data: the data reside in relational stores, and a knowledge graph is defined over it by a contextual layer of an ontology plus mappings. We call this *enterprise KGQA* to set it apart from KGQA over public graphs such as those QALD and LC-QuAD target: its vocabulary is proprietary and built by data engineers, and its users are domain experts rather than the schema's authors, so the distance between question and label defines the task instead of forming an edge case.

Enterprise data is moreover often spread across several providers, as in dataspaces, and this raises a question that a single-database task never poses: how two providers describe the same concept. Several providers describe overlapping concepts, and integration proceeds through agreed semantics rather than consolidation[18], so an ontology is the artifact that states which descriptions are equivalent. Both established routes rest on assumptions that may not hold here. Text2SQL grounds queries in table and column names[7;13;19], and those names do not themselves encode whether two providers mean the same thing[20]. If one records `wellbore_exploration` and another `exploration_boreholes`, the schemas do not express their equivalence (Appendix C). Materialized KGQA supplies that missing information through an ontology, but holds the data in a second copy that must be synchronized with the sources as they change. That conflicts with the governance arrangement behind federation[18] and answers from a snapshot rather than the current sources[21]. What remains is the semantics of KGQA without the materialization: ontology-based data access (OBDA) style R2RML/RML mappings serve as the semantic interface, and engines such as Ontop[22] rewrite SPARQL against the live sources. Deich et al.[23] framed natural-language querying on this basis as Virtual Knowledge Graph Question Answering (VKGQA).

In this paper, we therefore instantiate VKGQA as a minimal two-agent system, with a retrieval agent and a SPARQL generation agent. This brings enterprise KGQA to the agentic retrieval that recent KGQA and Text2SQL systems have adopted[10–12], but over mappings rather than a materialized graph. The retrieval stage is also what we manipulate: holding the LLM, the prompts, and the structural tools fixed, we vary only how the three linking tools match. The baseline is lexical substring matching over schema labels, which requires no preparation of the data. Against it we set embedding-based similarity, which requires embedding the schema and a sample of instance values in advance. We ask whether that preparation improves class, relation, and entity linking, and if so, under which conditions.

Our working hypothesis is that tool use mitigates the effect established for single-pass Text2SQL: if that effect is a linking failure rather than a generation failure (cf. Section 2.1), an agent that can search again where a single-pass model loses the term irrecoverably has exactly the capability those systems lacked. We test this across four input conditions, five datasets, and four LLM backbones, contrasting phrasings that merely corrupt the question string with those that replace the words denoting the schema, and we record what the agent does while searching alongside what it returns.

Measuring this requires an evaluation framework that current practice does not supply. Established KGQA benchmarks score against a single gold SPARQL query[14;15], which counts every other faithful reading of a question as wrong[24], and a result-level score cannot say whether a failure arose during retrieval or after it, which is the distinction between what a better matching mechanism can repair and what only the user can resolve. We therefore score against every admissible reading of a question and at three levels: the result, the schema the retrieval agent commits to, and the agent's search behavior (cf. Section 4).

This paper makes three contributions:

1. a corpus of enterprise KGQA questions in which the distance between question and schema vocabulary is varied under a paired design, with the ground truth held fixed;
2. a two-agent VKGQA system over R2RML/RML mappings whose retrieval stage can be swapped between lexical and embedding-based matching while everything else is held constant, together with the traces it produces;
3. an evaluation framework that scores a question against every faithful reading and attributes a failure to the stage of the pipeline that caused it.

The framework is general: it applies to any system evaluated on questions that admit more than one faithful answer. Federation is the setting that motivates the design. The experiments use five datasets, each a single source, and questions spanning providers are left to future work (cf. Section 7.3). We address three research questions:

(RQ1) How much result quality does a tool-using KGQA agent lose when a question is rephrased away from the schema vocabulary?

(RQ2) Does replacing lexical schema matching with embedding-based matching recover that loss?

(RQ3) Without explicitly being told to look out for underspecification, does the system report that a question cannot be answered as posed, rather than committing to a reading of its own, and does an explicit instruction change this?

2 Related Work

2.1 Robustness to Lexical Shift in Text-to-Query Systems

The closest antecedent of this work is Spider-Syn[5], introduced in Section 1. Its diagnosis is that schema linking in these models depends on lexical correspondence between question and schema tokens, and that questions displaying such correspondence dominate the benchmarks the field reports on. The overall accuracy loss is only part of the picture; two further findings of that study shape our design. Measured in exact-match accuracy on Spider versus Spider-Syn, the loss is steepest for the weakest models (GNN : $48.5\% \rightarrow 23.6\%$) and least for the strongest (RAT-SQL$_B$[2]: $69.7\% \rightarrow 48.2\%$), so capability mitigates but does not remove it. And a component-level breakdown shows where the loss falls: the parts of the query that name schema elements degrade sharply, while the rest of the query structure is almost unaffected. The failure is thus one of locating the right schema element, not of understanding the question. We generalize this attribution of a failure to a stage with retrieval metrics (cf. Section 4.2), which place the same question one level higher, between retrieval and generation rather than within a single query. Further benchmark variants study related problems. Spider-Realistic[25] removes explicit column-name mentions. Dr. Spider[26] extends this into a diagnostic benchmark of seventeen perturbation types across question, database, and SQL. Wang et al.[27] target ambiguous and unanswerable questions and find that models produce confidently wrong queries rather than acknowledging underspecification. Wretblad et al.[28] quantify how noise in BIRD-Bench distorts reported accuracies. For tabular NL interfaces more broadly, Gomm et al.[29] argue that conflating cleanly specified and ambiguous questions is a systematic blind spot of current evaluations. They separate the ambiguity a system may reasonably resolve on its own from queries so underspecified that no interpretation is recoverable, a distinction we adopt in Section 5.2.

PRACTIQ[30] makes ambiguity and unanswerability explicit in conversational Text2SQL, distinguishing four categories of each and constructing interactions in which the system may seek clarification before producing SQL. Its *ambiguous filter criteria* cases are particularly close to our UNDER condition: they are built by replacing a filter value with a relative term such as *large* or *young*, so the question leaves a decision open that schema lookup alone cannot settle. Our focus is narrower. Rather than evaluating the subsequent clarification dialogue, we ask whether a tool-using KGQA system recognizes such underspecification before answering at all, and separate that recognition failure from retrieval failures induced by vocabulary mismatch.

Our corpus construction follows the principle established by this literature: perturb the question, hold the ground truth fixed, and use ordinary rather than adversarial substitutions. Beyond the difference in system class already noted, two further differences separate the present study. Their remedies, multi-annotation selection and adversarial training, presuppose access to model training, which is unavailable in an LLM-agent setting without fine-tuning. More recent work shows that the robustness problem persists in LLM-based Text2SQL and addresses it the same way. Liu et al.[31] propose Solid-SQL, which combines a schema linker trained with LLM-augmented data with a two-round example-retrieval strategy, and report improved robustness on Spider-Syn, Spider-Realistic, and Dr. Spider. Without access to training, the corresponding lever is the retrieval mechanism, which we vary. And the setting differs: a single database with one schema whose labels are public, versus proprietary vocabularies over distributed sources that are never materialized.

On the KGQA side the gap is documented but never controlled. The survey of Steinmetz and Sattler[6] obtained its figures by running entity and relation linking tools over each question and checking which of the entities and properties the gold SPARQL query needs never appear among the candidates those tools return. Decisive for the present study is that all three linkers are single-shot: each returns one candidate list per surface form, and a required element absent from that list counts as missed. An agent is not bound to that verdict, since a lookup that returns nothing can be followed by another, phrased differently. Such a survey leaves open whether the gap these tools record still binds a system that can search repeatedly. It is why we treat the distance as an independent variable and record what the agent does while crossing it. Multilingual challenges such as QALD-10[14] and the Multilingual Text2SPARQL Challenge[32] introduce surface variation across languages but leave intra-language vocabulary divergence untouched.

2.2 Quality and Reliability of KGQA Benchmarks

A parallel line of work questions whether KGQA benchmarks measure what they claim. Zhang et al.[24] manually audited 16 widely used KGQA datasets and report an average factual correctness of 57%, with WebQSP at 52% and CWQ at 49.3%. The recurring defects are inaccurate or outdated annotations, trivially simple or unanswerable questions, and, most relevant here, rigid exact-match scoring that penalizes semantically correct answers expressed in a different surface form. Our adaptive ground truth (cf. Section 4.1) shares the premise that a single gold answer is too narrow a target, but extends it from surface form to ambiguity: a question may admit several readings with genuinely different result sets, and the scoring must accommodate all of them. Their critique concerns annotation quality and scoring. Ours concerns which questions benchmarks contain in the first place, and specifically that the lexical distance between question and schema vocabulary is left uncontrolled. The two critiques are orthogonal and mutually reinforcing.

2.3 Text2SPARQL

Translating natural language into SPARQL has long been studied in the context of KGQA, with benchmarks such as LCQuAD 2.0[15] and the QALD series[14;33] establishing DBpedia and Wikidata as common evaluation targets. Recent LLM-based methods such as SPARKLE[34] integrate the underlying knowledge graph directly into the decoding process, while SGPT[35] treats SPARQL synthesis as a graph-aware generation task. D'Abramo et al.[36] show that in-context learning alone, with retrieved demonstrations and

several generated hypotheses per question, reaches state-of-the-art results on these benchmarks without fine-tuning. These approaches[32;34–36] rely on a materialized graph as their primary input and therefore assume a fixed schema for planning and grounding.

The same assumption underlies recent agentic KGQA work, which nevertheless shares one property with our system: it retrieves iteratively rather than in one pass. BYOKG-RAG[10] lets an LLM propose entities, relation paths, and graph queries that specialized tools link to the graph, and refines its retrieval on what comes back. Its objective is robust context acquisition across custom graphs. RJE[11] retrieves reasoning paths, judges whether they suffice, and explores further only when they do not, which turns retrieval sufficiency into an explicit control signal. It optimizes answer-oriented reasoning over paths and uses predefined SPARQL for exploration. AutoLink[12] casts schema linking itself as an agent that expands a schema subset step by step. None of the three isolates the relation between question wording and schema vocabulary, and none targets a graph that exists only virtually, where the semantic interface is constituted by mappings over decentralized sources[37].

A second line of work addresses Text2SPARQL on top of OBDA infrastructures, where mappings rather than materialized graphs serve as the semantic interface. Emonet et al.[38] as well as Gómez and Fillottrani[39] examined how LLMs can extend OBDA frameworks with NL access, although their evaluations remained preliminary in scope. Approaches that rely solely on ontology classes as LLM input, such as the system of Askar et al.[40], are similarly limited, as they decouple query generation from the actual source-level mappings and therefore cannot reliably target specific data sources. Deich et al.[23] introduced VKGQA as an R2RML mapping-driven framing of Text2SPARQL for dataspace settings, in which transformed R2RML mappings serve as the semantic interface, and query execution is delegated to OBDA engines such as Ontop[22].

The line of work closest to ours in setting is that of Sequeda, Allemang, and Jacob. They established the benchmark case for enterprise KGQA[17], reporting that question answering over a knowledge graph defined by an ontology and mappings reaches 54% accuracy where direct SQL generation reaches 16%. Validating the generated SPARQL against the ontology and repairing it where the check fails subsequently raised this to 72%[41]. Our generation agent adopts that generate-and-verify structure (cf. Section 3.5) rather than reinventing it.

One thing distinguishes the present work methodologically. Their system supplies the full ontology in the prompt, so schema availability is not a retrieval problem in their setup: the model can still select the wrong element, but it cannot fail to see the right one. A vocabulary distributed across providers grows with each provider that joins, and Spider 2.0[16] and AutoLink[12] show that schemas of realistic size exceed the context limit of the LLM. That limit is typically 32k to 128k tokens for the small open-weight models an enterprise can host itself, and even the largest commercial models, whose windows reach about one million tokens, lose accuracy as the prompt fills. We therefore make schema discovery an explicit stage with a swappable matching mechanism; this allows the comparison in Section 6.2 and makes the lexical gap measurable here rather than assumed away. The agentic systems above do search, but they search one artifact that holds schema and instances together. Ours searches a schema assembled from mappings, with the data reachable through a separate SPARQL endpoint, so linking a class and linking an individual become separate operations over separate spaces.

2.4 Text2SQL

The Text2SQL task has been advanced primarily by LLM-based methods evaluated on cross-domain benchmarks such as Spider[13] and BIRD[19], with CHESS[7] a prominent agentic instantiation whose implementation has been openly released. Section 1 noted why direct SQL generation needs an additional semantic layer in federation: table and column names do not state that two providers model the same concept, which R2RML/RML mappings make explicit at the ontology level (Appendix C quantifies the difference). A second limitation is that Text2SQL is tied to relational sources, whereas federated ecosystems often include CSV, JSON, XML, and NoSQL sources[42], which RML covers.

Spider 2.0[16] and AutoLink[12] move Text2SQL closer to our setting, the former by evaluating enterprise workflows over large schemas that require consulting metadata and documentation, the latter by reformulating schema linking as iterative, agent-driven exploration that expands a relevant schema subset instead of exposing the full schema to the LLM and scales to industrial schemas. Both strengthen the case for retrieval-based interfaces, and AutoLink is the closest Text2SQL counterpart to our retrieval stage. Neither controls the relation between question and schema vocabulary, the variable we vary while holding the agent architecture and the structural tools fixed.

3 Materials and Methods

3.1 Problem Setting and System Interface

As outlined in Section 1, we study a two-agent system with a retrieval agent and a SPARQL generation agent. Both have access to exactly two resources, shown together with the agents' access rights in Figure 1: a schema $\mathcal{S}$ and an endpoint layer $\mathcal{E}$. Stating both precisely also settles how the system relates to conventional KGQA.

$\mathcal{S}$ A *schema*: the terminological part of the knowledge graph, comprising its classes, its object properties with domain and range, and its data properties with domain and datatype. $\mathcal{S}$ carries no data instances. Where the graph itself holds a literal value, $\mathcal{S}$ records only that value's datatype. A class `Learner` with a label, a division, enrolments, and a mentor is thus represented as follows, in the schema-triple notation used throughout this paper (class, property, range class or datatype), which is not an RDF statement about an individual:

```
eduo:Learner eduo:label xsd:string ;
  eduo:belongsTo eduo:Division ;
  eduo:enrolledIn eduo:Course ;
  eduo:mentor eduo:Educator .
```

which states what may be asked of a learner without naming any learner.

$\mathcal{E}$ A *SPARQL endpoint layer*: one or more endpoints over the knowledge graph itself, each evaluating a query Q to its result $\mathrm{sol}(Q)$. For a materialized graph this is a triple store; for a virtual one it is an OBDA engine such as Ontop[22], which rewrites Q into queries against the underlying sources at query time.

Given a natural-language question q, the task is to produce a SPARQL query Q such that $\mathrm{sol}(Q)$ answers q. The two agents differ in the access they are granted:

$$\text{retrieval agent:} \quad (q,\ \mathcal{S},\ \mathcal{E}|_{\text{lookup}}) \longmapsto (T_q,\ C_q), \quad (1)$$

$$\text{generation agent:} \quad (q,\ T_q,\ C_q,\ \mathcal{E}) \longmapsto (Q,\ R). \quad (2)$$

Here $T_q \subseteq \mathcal{S}$ is the set of schema triples the retrieval agent commits to, C_q the accompanying set of concrete entity references it has resolved, and $R = \mathrm{sol}(Q)$ the execution result. The restriction $\mathcal{E}|_{\text{lookup}}$ denotes bounded access: the retrieval agent may query $\mathcal{E}$ only to look up whether and in what form a given value occurs for an already-identified class-property pair. This allows it to ground a value-bearing term of q in an actual individual. It may not issue arbitrary queries. The generation agent has unrestricted access to $\mathcal{E}$, since composing and verifying Q requires executing candidate queries in full.

Materialized and virtual graphs: In enterprise KGQA the ontology and the mappings are given, and the knowledge graph they define may either be materialized or left virtual and queried through rewriting. Neither (1) nor (2) refers to how $\mathcal{E}$ answers a query, so the agents are indifferent to that choice. It affects only where $\mathcal{S}$ comes from: a materialized graph yields it directly, whereas a virtual one has no instance layer to inspect, so $\mathcal{S}$ must be assembled from the ontology together with the mappings, as Section 3.3 describes. The system is thus a KGQA system that additionally works where no graph has been materialized.

3.2 Agent Decomposition and Coordination

Answering q combines two distinct subtasks: locating the fragment of the knowledge graph needed to formulate a query for q, and composing a SPARQL query over that fragment which respects the endpoint's expressivity. Each subtask is assigned to its own agent, (1) and (2), for two reasons: a single LLM would otherwise have to reason over thousands of schema tokens and SPARQL syntax at once, and conflating the phases would let retrieval errors pollute the generation context. The first subtask concerns the graph, not merely its schema, since a question naming a company or a date can only be answered once that value has been grounded in an instance the data actually contains. This is why the retrieval agent is given lookup access to $\mathcal{E}$ alongside $\mathcal{S}$.

The size of $\mathcal{S}$ is bounded by the vocabulary, not by the data, so $\mathcal{S}$ can be embedded in full. Embedding the graph itself does not scale to realistic instance volumes and is impossible where no graph is materialized. The agents therefore search $\mathcal{S}$.

The agents communicate through a shared, typed state object. Only typed outputs cross the boundary, so neither sees the other's intermediate reasoning. Coordination is itself agent-driven rather than fixed, since a static pipeline cannot recover from over- or under-retrieval. Neither agent decides where control goes next on its own, and neither calls the other. Instead, each ends its turn by declaring what state it is in, and a deterministic, rule-based router reads that declaration and dispatches accordingly (Figure 1). A rule-based router keeps the architecture to two agents and holds the control flow fixed across backbones and runs, so every difference between configurations arises in the agents and their tools alone. Four outcomes are distinguished, and each corresponds to one situation an agent can find itself in.

The router treats a retrieval turn as EMPTY when the agent's output contains no parseable schema triple. It is thus a property of the output that the router checks, not a declaration by the agent. On the first empty turn the router returns control to the retrieval agent, so that it can search again and differently. On the second it ends the run as unanswerable. The retrieval agent declares $\textsc{Commit}(T_q, C_q)$ once it is satisfied that it has assembled the schema fragment the question needs, which hands that fragment to the generation agent and ends the retrieval phase. The generation agent declares $\textsc{Pending}(Q)$ when its current query Q still needs work, which returns control to itself for another revision. It declares $\textsc{NeedTriple}(d)$ when it finds that a schema element it needs is missing from what retrieval supplied. This sends control back to the retrieval agent, carrying the description d of the missing element so that the second retrieval pass knows what to look for. It declares $\textsc{Done}(Q)$ when it commits to Q as the answer. The retrieval agent may instead declare UNANSWERABLE, which ends the run without a query. In the default configuration this option is offered to it only once it has exhausted its tool budget and is asked to conclude, and only for the case that it found no relevant schema elements. Neither agent receives any instruction concerning underspecified questions, and the generation agent has no such option. A second configuration, used for RQ3 only, appends one rule to the system prompt of each agent, verbatim:

```
If the question cannot be answered with
any query without first asking the user
for missing information, do not invent
that information. Output exactly:
UNDERSPECIFIED: <what you would have to
ask the user, in one sentence>
```

The generation agent receives the same rule with `UNDERSPECIFIED` as an additional status of its JSON output, and the retrieval agents' forced conclusion offers it as a third option next to the schema output and UNANSWERABLE. The rule names no kind of missing information, so that it does not describe the five UNDER questions (cf. Section 6.3). Each phase is bounded by its own iteration limit, so an agent that keeps producing EMPTY turns or declaring $\textsc{Pending}(Q)$ terminates rather than looping indefinitely.

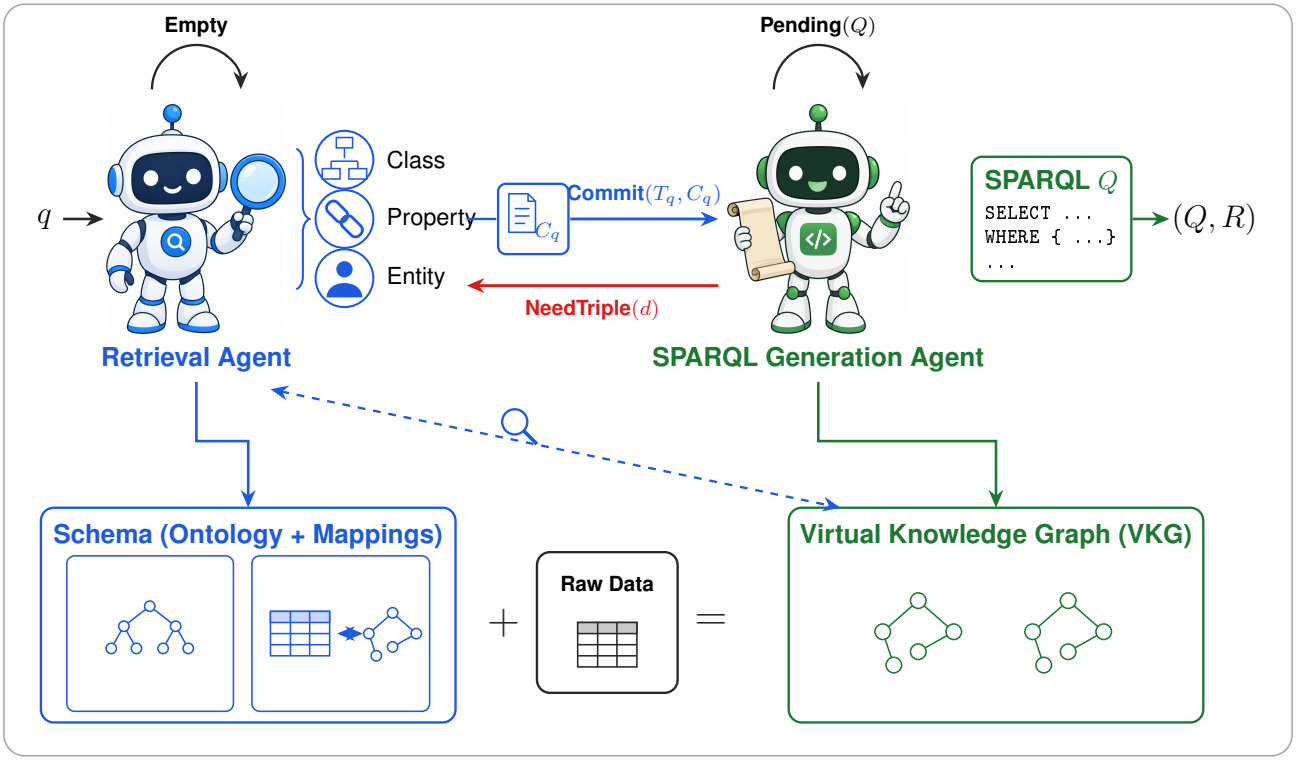

Figure 1. Sources, access rights, and status signals of the two-agent system. Dashed: the retrieval agent's bounded lookup access to the endpoint.

3.3 Constructing the Schema for a Virtual Graph

For a virtual knowledge graph, $\mathcal{S}$ cannot be read off a materialized instance layer, and it also cannot be taken to be the ontology alone. Neither source is sufficient by itself, and the reason is not incidental to the five datasets of this study (EDU, TRN, NRG, BSBM, and LCA, introduced in Section 5.1) but intrinsic to how OBDA works.

An ontology alone does not contain the information an OBDA engine needs to answer queries over relational data[22]. Only the R2RML/RML mappings establish the correspondence between ontology terms and source structures, and only they reveal which of the classes and properties the ontology declares are in fact populated with data instances, which an agent must know before writing a query against them. In practice the ontologies of these datasets are also incomplete as schema descriptions in their own right: properties, domains, ranges, or datatype declarations are variously missing, and where present they sometimes deviate from what the mappings produce. The mappings supply exactly this information.

The ontology is nonetheless indispensable, since the mappings describe only what is materialized from a source and therefore omit terminology that exists at the conceptual level only. Superclasses are the typical case: a question about *vehicles* must be answerable even when the mappings expose only `Car` and `Truck`, and only the ontology records that both are vehicles. We therefore construct $\mathcal{S}$ from both artifacts, and it is this construction, rather than the agents or their tools, that constitutes the difference to the materialized case.

Raw R2RML/RML documents carry substantial template, column, and datatype boilerplate that is irrelevant to schema reasoning and would inflate the agents' context budget. Following the mapping-transformation idea of Deich et al.[23], a pre-processing step therefore reduces each mapping document to schema-level triple patterns: classes, data properties with their datatypes, and object properties with their target classes, as in the `Learner` example above.

We extend that transformation in two respects. First, since OBDA engines accept a range of mapping languages and the original parser covers only a subset of R2RML, we replace static parsing with *minimal materialization*. The extractor loads the first 100 rows of each table, in dump order, into a database stub, runs a standard materialization engine over the unmodified mapping, and reads the schema off the resulting RDF. Templates, datatypes, and join conditions are thus instantiated with the source's own values rather than interpreted by a parser of our own (we use Morph-KGC[21] over an SQLite stub; the construction consumes only the RDF the engine emits and has not been evaluated with other engines). Second, the extractor consumes the OWL ontology and derives schema entries for the superclasses discussed above, aggregating the properties of their mapped subclasses so that a question posed at the conceptual level can still be answered. Appendix B gives the procedure step by step, together with the share of the mapped vocabulary it recovers for the five datasets.

3.4 Retrieval Agent

To produce (T_q, C_q) as in (1), the agent must solve two orthogonal subproblems: (i) identify which schema elements

the terms in q refer to, and (ii) find how those elements connect. We provide three tools for each, six in total.

The first subproblem is *linking*: mapping a mention in q onto the element of the knowledge graph it denotes. A question can refer to three kinds of elements, and the corresponding subtasks are those established in the KGQA literature[3;4]. *Class linking* and *relation linking* resolve mentions to classes and to properties, and both operate on $\mathcal{S}$. *Entity linking* resolves a mention to an individual in the data and is therefore the one subtask that must consult the graph itself; the bounded access $\mathcal{E}|_{\text{lookup}}$ provides this. Its output is C_q. We expose one tool per subtask, and this triplet is the interface on which our two configurations diverge.

Each tool has two interchangeable variants with identical input formats but different matching mechanisms. The *lexical* variant matches the search term as a case-insensitive substring of the local name of a class or property and returns every match, so it returns nothing when no label contains the term. For entity linking it issues a SPARQL `FILTER` over the values of the bound class-property pair. The *semantic* variant embeds the search term with the same model as the schema (text-embedding-3-large) and ranks the elements of the requested type by cosine similarity over an index built once per dataset. The text embedded for an element is its local name with camel case split into words, for a property followed by its domain class (*activeDepositOperator active Deposit Operator of Deposit*). It returns the k nearest elements, with a default of $k = 5$ that the agent may raise per call, applies no similarity threshold, and breaks ties by index order. For entity linking it embeds the first 500 distinct values of the class-property pair, in the endpoint's fixed order, and returns the 50 nearest to the mention; only one pair in our datasets exceeds that window (531 company designations), and every entity a corpus question names lies within it. Besides the similarity function, the variants therefore differ in one further respect: a lexical search can return nothing, whereas a semantic search over classes or properties always returns k candidates. The search-failure rate of Section 4.2 is consequently informative for the lexical variant only. For the semantic variant the question is whether the right element is among the k, which the retrieval metrics answer. The semantic variant exists because lexical matching must fail wherever a mention and a label share a meaning but not a surface form, the *lexical gap* of Section 2.1, which embedding similarity should bridge.

The second subproblem is navigating the schema once entry points have been linked. Three needs recur: inspecting a class's neighborhood, following the class hierarchy, and connecting classes that share no direct property. One tool addresses each, returning respectively all triples of a class, its `rdfs:subClassOf` hierarchy, and the shortest property path between two classes. These structural tools are identical in both configurations, as are the LLM and the prompts, so any difference between the configurations arises in the three linking tools: in their similarity function and, as noted above, in whether a search can come back empty. Before handing over, the router checks that T_q parses as a connected RDF graph.

3.5 SPARQL Generation Agent

The generation agent receives q, the retrieved triples T_q, the entity references C_q, and the endpoint associated with T_q.

Because answer correctness cannot be established from the query text alone, the agent operates in a generate-execute-revise loop. It produces a candidate Q, the router executes it and returns a summary on the next turn, and the agent then revises Q, requests a missing schema element, or commits. The summary reports the first result bindings on success, the error class on failure, or the OBDA-reformulated SQL on timeout. This is the same generate-and-verify principle that Allemang and Sequeda[41] established, and the comparison is instructive because their setting is otherwise close to ours: they too query a virtualized knowledge graph over mappings rather than a materialized one. Two differences matter. Their verification is an ontology-based query check, a deterministic rule set over the ontology that inspects the query in isolation and is by design not an LLM. Ours is an LLM applying a verification checklist to the query *together with its execution result*, which admits errors that no static check detects, such as a well-formed query returning nothing. An empty result does not end the loop and is not treated as an error by rule. The agent is instructed to check the `FILTER` literal against values sampled from the data (the instance-value sampler below) before it decides, and it may then revise the query or commit to the empty result as the answer. Whether an empty result is accepted is thus a judgment of the agent, informed by the sampled values. And their repair loop is the whole of the system, whereas ours can hand control back to retrieval when the fault lies in T_q rather than in Q. How often this happens, and with what outcome, is reported with the *Retry* column of Section 6.1.

The agent has one auxiliary tool, an *instance-value sampler*, which returns up to ten distinct values for an already-bound class-property pair, taken in the endpoint's fixed order rather than at random. It is needed because a `FILTER` matches only if its literal agrees with the stored format and unit, which vary across sources. The sampler only shows the agent how values are stored: their spelling (*STATOIL* rather than *Statoil ASA*), their unit and scale (a depth in metres, a production volume in millions), and their range. It does not decide which class or property a mention refers to. Retrieval has settled that before generation starts.

Each turn ends with the current Q, a free-text reasoning, and one of the declarations of Section 3.2: PENDING(Q), NEEDTRIPLE(d), or DONE(Q). Self-revisions and NEEDTRIPLE round-trips are bounded separately; if either limit is reached without a DONE, the question is reported unanswerable.

4 Evaluation Framework

We evaluate at three levels: the result (cf. Section 4.1), the retrieved schema (cf. Section 4.2), and the agent's search behavior (cf. Section 4.3). A score on the result alone reports that a run failed, but not where it failed. The other two levels supply that. If the agent never retrieved the right schema, the fault lies in retrieval and a better matching mechanism can address it. If it retrieved the right schema and still answered wrongly, the fault lies elsewhere and better retrieval will not help. The input conditions of Section 6 differ in which

of the two holds. The framework is independent of the system described in Section 3, presuming only a question, the system's query together with its result set, and annotated ground truth queries.

4.1 Result Evaluation

A correct answer to an ambiguous question may legitimately vary along two independent axes, and the adaptive ground truth is built to accommodate both.

The first is *interpretation ambiguity*: the question admits several readings that yield genuinely different result sets. Consider, for example, "boreholes with a total drill depth greater than 3000 m" posed against our oil and gas dataset, whose schema exposes two depth properties: `eno:totalDrillDepth` measures the borehole along its path, `eno:finalVerticalDrillDepth` how far down it reaches. Both are defensible readings, and they select different boreholes. Rather than declaring one of them correct, we enumerate one ground truth query per reading and credit the agent for matching any of them.

The second is *granularity ambiguity*: the readings agree on which rows to return but differ in how many columns a reasonable answer projects. A query for boreholes might return identifiers alone, or identifiers together with their human-readable designations. The latter is more helpful and no less correct, so counting the extra column as a false positive would penalize helpfulness. We therefore annotate columns as required or acceptable and let only the former drive recall. Separating the two axes allows a single score to remain meaningful under ambiguity.

Adaptive ground truth: For each NL query q, we manually construct a set $\mathcal{Q}_q = \{Q_1, \ldots, Q_k\}$ of admissible ground truth (GT) SPARQL queries, in which each Q_i encodes one valid interpretation of q. The adaptive ground truth is then the family of admissible solutions

$$G_q \;:=\; \{\, \mathrm{sol}(Q) : Q \in \mathcal{Q}_q \,\}, \tag{3}$$

where $\mathrm{sol}(Q)$ denotes the set of distinct result tuples that Q produces from the source data. Duplicate rows, which SPARQL may return, are removed on both sides before scoring, so a row counts once however often it is produced. A system answer is scored against each member separately and reported against the member it matches best (Equation 6). The members are never pooled. The readings and the required/acceptable labels were authored by the first author. They arise along a small number of systematic dimensions rather than from free interpretation: with or without explicit type assertions, aggregated or not, with or without `DISTINCT`, and, for one question, an alternative measure. Of the 17 paired questions, 2 have one reading, 10 two, 4 four, and 1 ten. Every reading was executed against the endpoints, and all readings and labels are part of the released material, so the annotation can be inspected and contested reading by reading.

Two result tables are compared: L_q, obtained by executing the query the system committed to, and $\mathrm{sol}(Q)$ for a candidate Q. We write $\mathrm{cols}(X)$ for the columns of a table X and $X[c]$ for the set of distinct values in column c, and call the two kinds *system columns* and *GT columns*. Their number, names, and order need not agree; the matching below resolves this.

Every GT column c carries two annotations: a *semantic concept* (SC) $\sigma(c)$ and a *relevance level* $\ell(c) \in \{\textsc{required}, \textsc{acceptable}\}$, the latter realizing the granularity axis above. The SC names the thing the question asks about, stated in the terms of the question rather than of the schema. We say concept rather than class because an SC may be realized by a class, a property, or a literal column, whichever the mappings happen to use. A question about learners and the courses they take has the SCs *learner* and *course*, whatever the mappings happen to call them. Several columns may realize one SC: the query typically returns both a URI and a label per entity, and both carry the same $\sigma(c)$, so producing either counts as addressing that concept. This allows us to ask whether the agent answered the right question, independently of how it chose to render the answer. Because joining an `ACCEPTABLE` column as a required pattern would discard otherwise-valid rows in which only that attribute is missing, we wrap such columns in `OPTIONAL` in the GT SPARQL, so their values surface in $\mathrm{sol}(Q)$ without restricting the row set induced by the `REQUIRED` columns.

Matching system and GT columns: Since column names need not agree, columns are matched by the values they contain, separately for each candidate $Q \in \mathcal{Q}_q$. We write $\nu(X[c])$ for the *non-trivial* values of a column: the values after Unicode normalization, whitespace stripping, and lower-casing, excluding zeros, empty strings, booleans, and the common null placeholders. The overlap of a system column c' and a GT column c is

$$\omega_Q(c', c) \;=\; \big|\, \nu(L_q[c']) \,\cap\, \nu(\mathrm{sol}(Q)[c]) \,\big|, \tag{4}$$

set to 0 for columns whose shared values are all numeric or all temporal unless c and c' stem from the same SPARQL property, since two such columns may share digits without representing the same quantity. The property of a system column is read from the system's query: from the triple pattern that binds the variable or, for an aggregate, from the patterns of the aggregated variable. A framework applied to a system that exposes only result sets loses this safeguard and pairs numeric columns by values alone. We use the absolute count rather than a normalized similarity or a threshold. A column with many shared values must outrank one with few, and a threshold would penalize questions whose correct answer has only a handful of rows. A coincidental overlap on a single shared value can therefore establish a pairing only where no larger overlap competes for either column. Each system column is paired with the GT column it overlaps most, greedily in descending order of overlap and never using a column twice. Ties are broken by GT column order and then by the order of the system's projection. The result is a partial injective map μ_Q from system to GT columns. It is partial because columns on either side may remain unpaired, and the metrics measure this. We verified the greedy pairing against a maximum-weight bipartite matching for all 6,674 pairings computed in this study: the total overlap is identical in every case, and the pairings differ only in 114 cases in which two system columns tie for the same GT column.

Schema metrics and candidate selection: On this pairing we build two families of metrics, which separate

missing a required concept entirely from addressing it with the wrong query body. *Schema metrics* score whether the agent's `SELECT` clause projects the concepts the question asks about. *Data metrics* score whether the rows beneath those concepts are correct.

Schema metrics ask whether the system addressed the semantic concepts the question asks about. Precision is the share of its columns that found a partner, recall the share of the REQUIRED concepts of Q that a partnered column realizes:

$$P_{\text{schema}}(Q) = \frac{\#\,\text{paired system columns}}{\#\,\text{system columns}},$$
$$R_{\text{schema}}(Q) = \frac{\#\,\text{covered REQUIRED concepts of } Q}{\#\,\text{REQUIRED concepts of } Q}, \quad (5)$$

where a REQUIRED concept counts as covered if at least one GT column carrying it was paired. Recall counts concepts rather than columns because a concept is usually represented by two GT columns, an identifier and a label (262 of the 361 concepts in our ground truth), and a system that projects either one has addressed the concept. Precision counts columns because every system column without a partner is a column the user receives without warrant, whichever concept it misses, and the system does not declare which concept an unpaired column was meant to realize. A system that projects both identifier and label of a concept is not penalized, since the GT carries both columns and both are paired. Note that an ACCEPTABLE column raises P_{schema} when paired but is never required, so producing it is rewarded and omitting it is not penalized. We write $F_{1,\text{schema}}$ for the harmonic mean of the two. The candidate

$$Q^* = \arg\max_{Q \in \mathcal{Q}_q} \left(R_{\text{schema}}(Q),\ F_{1,\text{data}}(Q),\ P_{\text{schema}}(Q)\right) \quad (6)$$

is taken as the matching GT solution, where the maximum is lexicographic. It selects the reading whose REQUIRED concepts the system covers best, among those the one whose rows it matches best (the data metrics are defined below), and only then the one with the higher schema precision. Recall decides first because it depends only on what the system addressed, whereas precision depends on which ACCEPTABLE columns a reading happens to list, and data F_1 would favor a reading whose REQUIRED concepts the system misses, since the disagreeing column then drops out of the row comparison (Q_2 in Figure 2).

For Q^*, *data metrics* compare the rows. Both tables are projected onto the paired columns, with μ_{Q^*} fixing which column is compared against which, and precision, recall, and F_1 are computed over the resulting sets of distinct tuples in the standard way. Figure 2 exemplifies this matching process, and Appendix A walks through the example step by step.

4.2 Retrieval Evaluation

We additionally evaluate the intermediate output of the retrieval agent in isolation. For each admissible query $Q \in \mathcal{Q}_q$, a retrieval ground truth R_Q is extracted by traversing its algebra tree and collecting the schema elements the agent should have surfaced: classes, properties, and instance references. Relevance levels carry over from Section 4.1, with elements confined to `OPTIONAL` blocks counting as ACCEPTABLE and all others as REQUIRED. Retrieval precedes the choice of interpretation. At that stage the agent cannot yet know which reading it will commit to, so surfacing the schema of any one admissible reading is correct behavior. The retrieved triples are therefore scored against each R_Q separately, as the result-level metrics score against each $\text{sol}(Q)$. Recall and path coherence are reported for the reading the agent covers best. Precision is measured against the union of all R_Q, so that an element belonging to any admissible reading is never counted as a false positive.

We then compare R_Q against the schema triples T_q that the retrieval agent commits to at the end of the retrieval phase. Matching primarily expects the GT class itself: a retrieved triple is a direct match when its subject equals the GT class of the corresponding GT triple. T_q may instead contain concrete subclasses of the GT class, following the ontology hierarchy lifted by the schema extraction step (cf. Section 3.3). In that case the GT triple is expanded into one expected triple per concrete subclass with mappings, and recall is measured over the expanded set. This way, retrieving only some of several relevant subclasses produces a proportional recall hit rather than full credit for partial coverage. Precision, recall, and F_1 over schema triples are computed in analogy to the result-level metrics, with the difference that ACCEPTABLE GT triples missing from T_q do not count as false negatives, and ACCEPTABLE triples present in T_q count as true positives.

We additionally report *path coherence*, which asks whether each property of R_Q was surfaced attached to its domain class, the anchor the generated query needs for its joins. Let E_Q be the (domain class, property) edges of R_Q, each labelled REQUIRED or ACCEPTABLE as above. For an edge $e = (C, p)$, let $m(e) = n(e) = 1$ if T_q contains a triple with subject C and predicate p. Otherwise, let S be the concrete subclasses of C that carry p in the schema, $n(e) = |S|$, and $m(e)$ the number of $s \in S$ for which T_q contains (s, p); when S is empty, $n(e) = 1$ and $m(e) = 0$. For an edge whose subject is untyped in Q, $m(e) = n(e) = 1$ if p occurs as a predicate in T_q and $m(e) = 0$ otherwise. Then

$$\text{PC}(T_q, R_Q) = \frac{\sum_{e \in E_Q} m(e)}{\sum_{e\ \text{REQUIRED}} n(e) + \sum_{e\ \text{ACCEPTABLE}} m(e)}, \quad (7)$$

so that a missing ACCEPTABLE edge does not lower the score and a partially retrieved subclass expansion counts proportionally. Unlike triple recall, path coherence ignores the range of the property, so the two metrics diverge exactly where the agent found the right class and property but the wrong range. Like recall, it is reported for the reading the agent covers best.

4.3 Tool-Use Metrics

The metrics above score what the agent produces. To ask whether tool access mitigates vocabulary mismatch, we additionally measure what the agent *does* while producing it. This is the behavior that single-pass models do not exhibit and that prior robustness studies therefore could not observe. From each trace we extract two quantities. The *search count* is the number of invocations of the three core retrieval tools per run, indicating how much effort the agent expends locating schema elements. The *search failure rate* is the fraction of those invocations that return no usable match,

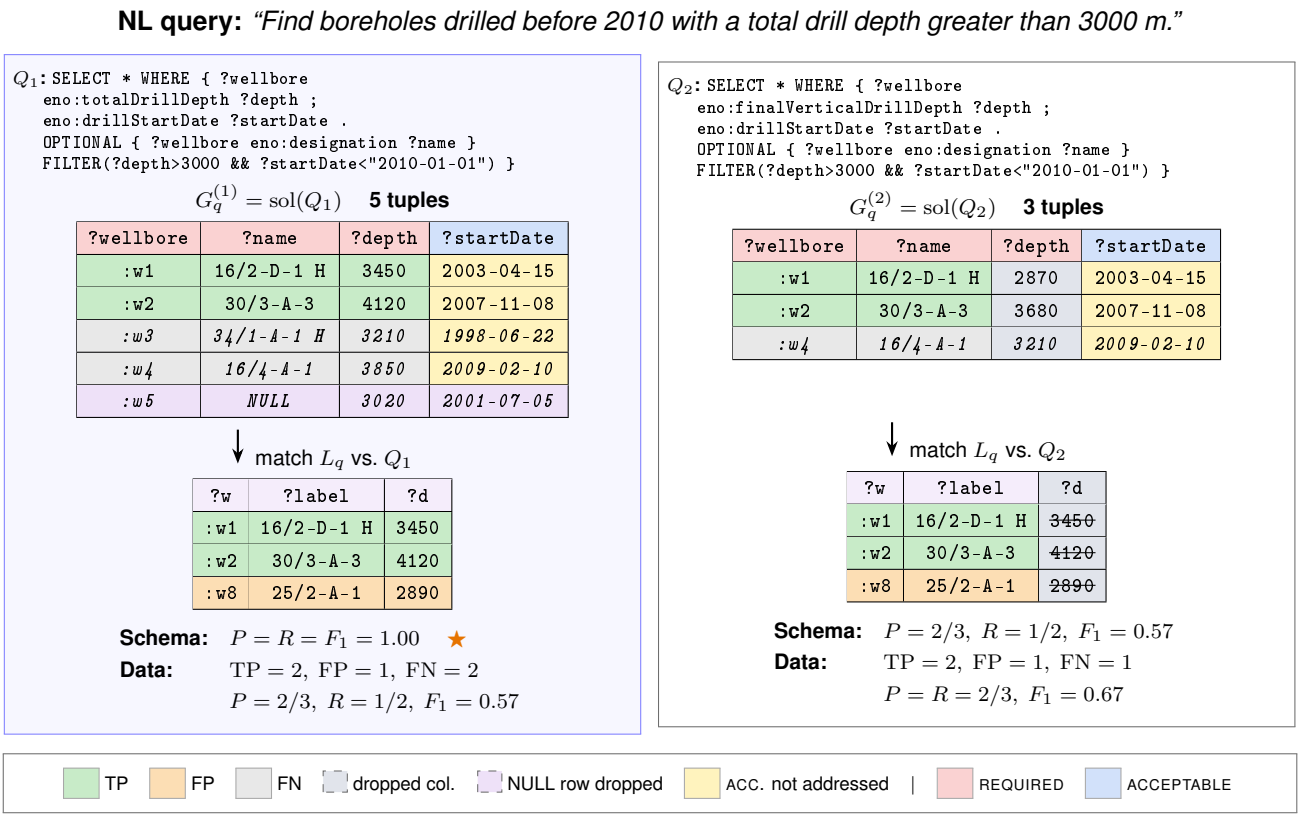

Figure 2. Result-level evaluation on the running example. L_q is matched against both GT solutions $\mathrm{sol}(Q_1)$ and $\mathrm{sol}(Q_2)$, and the candidate whose REQUIRED concepts the system covers best (Q_1, schema recall 1.00 against 0.50) is reported.

indicating whether that effort is productive. Together with the retrieval metrics of Section 4.2, these separate three states that a single result-level score conflates: the agent did not look, the agent looked and found nothing, or the agent looked, found the right schema, and still failed to answer.

5 Experimental Setup

5.1 Datasets

We evaluate on five publicly available datasets covering different domains and schema sizes: EDU (LUBM[43], university), TRN (GTFS[44], public transit), NRG (NPD[45], oil & gas), BSBM[46] (e-commerce), and LCA (life-cycle assessment, ORIENTING[47]). Per-dataset schema, mapping, source, and query-distribution statistics are reported in Table 1. The selection is constrained by what the scenario of Section 3.1 requires, namely all three of raw data, a domain ontology, and R2RML/RML mappings tying the two together. Most KGQA benchmarks ship a materialized graph without the sources it was derived from[6] and are therefore unusable here. We identified five public datasets that satisfy these requirements and provide sufficient QA tasks.

None of them is materialized: each is exposed as a virtual knowledge graph through a dedicated Ontop endpoint[22], so every candidate query is rewritten into SQL over the live sources at query time. This leaves the retrieval problem unchanged (cf. Section 3.1) but exposes the generation agent to realistic execution costs: a poorly constrained query can run for seconds or hit the endpoint's timeout.

5.2 Question Conditions

The corpus comprises 56 question formulations derived from 22 underlying intents: 17 paired intents with BASE, SYN, and TYPO variants and 5 additional UNDER intents. Each formulation belongs to one of four *conditions*, BASE, SYN, TYPO, and UNDER, which differ only in how the question is phrased relative to the schema vocabulary. BASE, SYN, and TYPO form a paired within-subject design. Each of the 17 questions appears in all three conditions with an *identical* ground truth, so that any difference in score is attributable to the phrasing alone and not to task difficulty. UNDER consists of 5 further questions that have no BASE counterpart by construction.

BASE (17) The question uses the vocabulary of the mappings verbatim. It serves as the control against which the other conditions are measured.

SYN (17) Schema-related terms are replaced by the wording a user without schema knowledge would plausibly choose. The corpus records each substitution with its kind: plain synonyms (*learners* → *students*), hypernyms (*learners* → *people*), circumlocutions (*stop events* → *halt*), format changes (*3000 meters* → *3 kilometers*), and two renamed entities (*Statoil* → *Equinor*, *Det norske oljeselskap* → *Aker BP*). As in Spider-Syn[5], substitutions are plausible user wording rather than adversarial choices, and the ground truth is unchanged. This condition isolates vocabulary mismatch; Section 6.2 shows that the residual loss

Table 1. Per-dataset statistics. *Classes* and *Properties* report the effective schema $\mathcal{S}$ available to the agents, assembled from the ontology together with the mappings (cf. Section 3.3). *Prop.* abbreviates properties and *Tables* the number of relational source tables. *Vocabulary* states whether the schema names the agents see were renamed for this study, left as published, or had not been published at the time of the experiments (cf. Section 5.2). Query counts refer to question formulations (56 in total, derived from 22 underlying intents as described in Section 5.2).

Dataset	Domain	Classes	Prop.	Triple Maps	Tables	Queries	Vocabulary
EDU	University	18	18	22	14	16	renamed
TRN	Transit	12	44	13	10	12	renamed
NRG	Oil & Gas	113	290	1177	73	19	renamed
BSBM	E-commerce	8	40	42	10	5	as published
LCA	LCA	15	15	14	2	4	unpublished

under semantic retrieval follows two properties of question and schema rather than the linguistic kind of the substitution.

TYPO (17) Entity or attribute references carry spelling errors, with vocabulary otherwise identical to BASE. This separates surface-level noise from genuine vocabulary mismatch: both perturb the question string, but only SYN changes which words denote the schema.

UNDER (5) The question omits a criterion or a threshold that only the user can supply, as in "Show top products" (top by which measure, and how many?) or "Find large departments" (large by which count, above which cutoff?). These are what Gomm et al.[29] call *uncooperative* queries: ambiguous in the question itself.

UNDER differs structurally from the paired conditions. Those hold the intent of a question fixed and vary only its wording, so a faithful query always exists and what changes is how hard its schema elements are to find. Where such a question admits several readings, the ambiguity is one of grounding: it arises in the schema, not in the question, and the adaptive ground truth credits each reading (cf. Section 4.1). An UNDER question is ambiguous before any schema is consulted. It leaves open a decision that no vocabulary could settle, so no gold query exists and no rephrasing links it to a BASE counterpart, which is why the condition is unpaired and smaller. It is scored on a single behavioral criterion instead: whether the system reports the question as underspecified rather than committing to a reading of its own (cf. Section 6.3). In a user-facing system the appropriate response would be a clarification question; recognizing that one is needed is its precondition and the part that can be scored without a user in the loop, so the report stands in for the clarification turn in this study.

Controlling for memorization: In general, enterprise vocabularies are proprietary and authored by the organization's data engineers (cf. Section 1). It is important to note that our datasets are public, which is where they diverge from the setting they represent. The original mappings and ontologies of LUBM, BSBM, NPD, and GTFS are widely published and therefore likely present in the LLMs' training data. This is the benchmark-contamination problem that Sainz et al.[48] argue must be controlled for per benchmark rather than assumed away. A model that reproduces a memorized `Student` class from LUBM has not performed schema retrieval at all. BASE would then measure recall of training data rather than vocabulary-faithful querying, and the BASE/SYN contrast would collapse.

We therefore rewrote the identifiers of EDU, TRN, and NRG with a scripted rename table applied systematically to the ontologies, the mappings, and the gold queries. 45 classes and 97 properties received new local names and the namespace prefixes were replaced, so that LUBM's `Student` and `Department` become `Learner` and `Division`. The ontologies carry no `rdfs:label`, so the local name is the only name an agent sees. Source table and column names, instance data, and the graph structure are unchanged. BSBM was left as published because its vocabulary consists of generic commerce terms (`Product`, `Vendor`, `Offer`) of the kind the rename also leaves in place elsewhere. LCA needed no rename either: its ontology and mappings were authored for the ORIENTING work[47], which had not been published when the experiments were run, so the vocabulary the agents see cannot be in any training corpus. Of the 64 schema elements the BASE questions require, 52 carry a rewritten name. The remaining 12 are generic terms such as `Course`, `Stop`, `Trip`, `Company`, and `price`. None of the original identifiers occurs in the retrieval index. BASE questions use the rewritten names. SYN questions use the terms a user would be expected to produce, which are the original ones ("student", "department"). The rename thus reduces direct memorization of schema identifiers, as the BASE/SYN contrast requires.

5.3 System Configuration and LLMs

The two retrieval agents and the SPARQL generation agent share the same underlying LLM within a run. We evaluate four models spanning vendors, access regimes (closed-source vs. open-weights), and capability tiers (frontier reasoning at two generations, frontier MoE, mid-size dense), chosen to assess robustness across model families: *DeepSeek-V3.2* (December 2025 release, API access), *GPT-5.4* and *GPT-6.1 Sol* (OpenAI, reasoning effort `medium`; the latter is the current frontier model at the time of writing), and *Qwen3.5-27B-Instruct* (via OpenRouter). Sampling temperature is fixed to 0.1 for DeepSeek-V3.2 and Qwen3.5-27B-Instruct. The retrieval and SPARQL per-phase limits (cf. Section 3) are set to 3 each to balance recovery from generation errors against inference cost. Furthermore, embeddings for the semantic configuration are computed

with OpenAI `text-embedding-3-large` and cached per dataset.

Run protocol: The experimental unit is the question: 17 paired intents for BASE, SYN, and TYPO and 5 for UNDER. For every (model, configuration, question) triple we execute six replicate runs to absorb the residual stochasticity of tool-use branching. The replicates are averaged per question before any statistic is computed, so that every mean and every error bar reported below is taken over questions, not over runs. This yields 102 traces per cell for BASE, SYN, and TYPO and 30 for UNDER, in each of the two retrieval configurations and for each of the four LLMs. Each run is recorded as a structured trace covering message history, tool-call log, candidate SPARQL queries, and execution results. Every figure reported below derives from these traces, and the analysis script is part of the released material.

6 Results

6.1 Vocabulary Mismatch (RQ1)

RQ1 asks how much result quality is lost when the question's wording departs from the schema vocabulary. Under the lexical retrieval interface, the effect reported for Text2SQL reproduces clearly. Moving from BASE to SYN reduces data F_1 by 0.220 for GPT-5.4, by 0.234 for GPT-6.1 Sol, by 0.294 for DeepSeek-V3.2, and by 0.303 for Qwen3.5-27B, a loss of 22 to 30 points. The question differs only in word choice, and its ground truth is identical. The loss is of the same order as the 20 to 30 points that Gan et al.[5] report for Spider-Syn, obtained on a different task, a different query language, and a different class of system, so the two numbers indicate a comparable magnitude rather than a direct replication. That the effect appears at all in tool-using agents over non-materialized mappings is the first finding: under a lexical retrieval interface, the ability to search and revise does not by itself make a system robust to vocabulary mismatch.

Figure 3 shows the pattern per condition and backbone. The four models track each other closely: the ordering of the conditions is identical for all of them, and the SYN drop is of comparable size in each, suggesting that the effect is not confined to one model family.

The conditions do not, however, behave alike, and the differences are more informative than the aggregate. Under TYPO the result quality remains at the BASE level. Its data F_1 lies within 0.036 of BASE for GPT-5.4 (0.827 vs. 0.791), within 0.007 for GPT-6.1 Sol, within 0.013 for DeepSeek, and within 0.014 for Qwen. The tool-use columns of Table 2 explain why: search counts and failure rates under TYPO match BASE almost exactly (6.9 vs. 6.5 searches and 19.4% vs. 19.6% failures for GPT-5.4). Misspelled schema terms rarely reach the tool layer: across the 816 TYPO runs, a misspelled token appears verbatim in 96 of 3,870 linking-tool calls (2.5%). Most of these (74) are misspelled entity names passed to the value search (*trol*, *oljeslskap*), where the linker resolves them against the data. The LLM normalizes misspelled schema terms in context before issuing a call. Surface perturbation of the question string therefore does not reduce performance; only perturbation of *which words denote the schema* does.

Under lexical retrieval, SYN raises the search count over BASE by 78% for GPT-5.4 (6.5 → 11.6), 47% for GPT-6.1 Sol (4.3 → 6.3), 63% for DeepSeek (4.8 → 7.8), and 69% for Qwen (5.5 → 9.3). Insufficient search activity therefore does not explain the loss. The agents plainly detect that something is wrong and respond by searching more. The searches, however, are unproductive: the failure rate more than doubles in every case, from 19.6% to 43.1%, from 14.1% to 37.4%, from 20.1% to 42.9%, and from 17.0% to 44.4% respectively. Four independent models converge on a similar pair of numbers. Class and relation linking are performed by substring matching, which often cannot resolve a mention that means the same as the label but is not a substring of it.

The effect is a degradation rather than a block. A majority of searches still succeed under SYN, and the schema retrieved remains far from empty, with triple F_1 at 0.711 against a BASE level of 0.805 for GPT-5.4. Entity linking, which queries the graph rather than the labels, is unaffected throughout, which is why the failure is specific to schema linking rather than general. What the added effort does not do is close the gap, which is the observation that single-pass architectures cannot yield: the capacity to retry is not the capacity to recover.

A third behavioral signal confirms this from the level of the retrieval phase as a whole. The *Retry* column records how often the generation agent sent control back to retrieval after an unsatisfactory first attempt. Under lexical retrieval it rises from BASE to SYN for every model, most sharply for GPT-5.4 (10.8% to 38.2%) and proportionally even more for Qwen (2.0% to 17.6%), while TYPO stays at the BASE level throughout. The agents are thus restarting the retrieval phase itself, the strongest available indication that they detect the failure rather than terminating in false confidence. The restart seldom rescues the run: runs that restarted retrieval reached a mean data F_1 of 0.07 against 0.81 for those that did not, and fewer than one in twenty of them ended above 0.9. All in all, RQ1 is answered as follows: under lexical retrieval, vocabulary mismatch reduces data F_1 by 22 to 30 points, misspellings leave it unchanged, and the agents detect the mismatch and search more without recovering from it. Whether a different matching mechanism recovers it is the question RQ2 addresses.

6.2 Embedding-Based Matching (RQ2)

RQ2 asks whether embedding-based matching recovers that loss. Swapping only the three core retrieval tools for their embedding-based variants, holding the LLM, the prompts, and all structural tools fixed, removes the failures partly. Under SYN, the search-failure rate falls from about two fifths to a few percent for all four models, search counts fall in step, and the generation agent hands control back to retrieval less often, for DeepSeek not at all (Table 2). The elevated effort under lexical retrieval was thus compensation for unproductive lookups rather than intrinsic task difficulty. The result quality follows for three backbones: SYN data F_1 rises from 0.571 to 0.686 for GPT-5.4, from 0.528 to 0.641 for DeepSeek, and from 0.507 to 0.601 for Qwen (Figure 4). For GPT-6.1 Sol it falls from 0.575 to 0.543. The retrieval metrics show that the gain originates in retrieval:

Table 2. Retrieval quality and agent behavior by input condition, retrieval configuration, and LLM. *lex* is the lexical (substring) interface, *sem* the embedding-based one. *trF1* is the schema-triple F_1 and *pCoh* the path-coherence score (cf. Section 4.2); *Srch* is the mean number of linking tool calls per run, *Fail* the fraction returning no usable match, and *Retry* the share of runs in which the generation agent handed control back to retrieval (NEEDTRIPLE, Section 3.2). Bold marks the values discussed in the text below. Result quality is reported in Figures 3 and 4.

		Retrieval		Agent behavior		
Set	**Cfg**	**trF1**	**pCoh**	**Srch**	**Fail**	**Retry**
GPT-5.4						
BASE	lex	0.805	0.901	**6.5**	**19.6%**	**10.8%**
SYN	lex	0.711	0.820	**11.6**	**43.1%**	**38.2%**
TYPO	lex	0.836	0.937	**6.9**	**19.4%**	9.8%
BASE	sem	0.843	0.941	4.8	5.2%	1.0%
SYN	sem	0.697	0.857	**6.8**	**4.5%**	**18.6%**
TYPO	sem	0.805	0.946	5.0	6.3%	5.9%
GPT-6.1 Sol						
BASE	lex	0.863	0.947	4.3	14.1%	11.8%
SYN	lex	0.834	0.903	**6.3**	**37.4%**	**28.4%**
TYPO	lex	0.839	0.962	4.0	14.3%	11.8%
BASE	sem	0.879	0.956	3.3	2.8%	2.9%
SYN	sem	**0.715**	**0.823**	5.5	**8.0%**	**22.5%**
TYPO	sem	0.865	0.968	3.4	4.8%	5.9%
DeepSeek-V3.2						
BASE	lex	0.817	0.848	**4.8**	**20.1%**	0.0%
SYN	lex	**0.578**	**0.641**	**7.8**	**42.9%**	7.8%
TYPO	lex	0.789	0.853	5.1	20.9%	1.0%
BASE	sem	0.821	0.859	4.3	0.0%	0.0%
SYN	sem	**0.661**	**0.762**	5.8	**2.0%**	**0.0%**
TYPO	sem	0.869	0.930	4.5	0.0%	0.0%
Qwen3.5-27B						
BASE	lex	0.647	0.661	**5.5**	**17.0%**	**2.0%**
SYN	lex	0.605	0.646	**9.3**	**44.4%**	**17.6%**
TYPO	lex	0.640	0.641	5.0	19.0%	1.0%
BASE	sem	0.479	0.503	3.9	0.7%	0.0%
SYN	sem	0.585	0.675	6.2	**1.6%**	**4.9%**
TYPO	sem	0.539	0.595	4.1	0.0%	0.0%

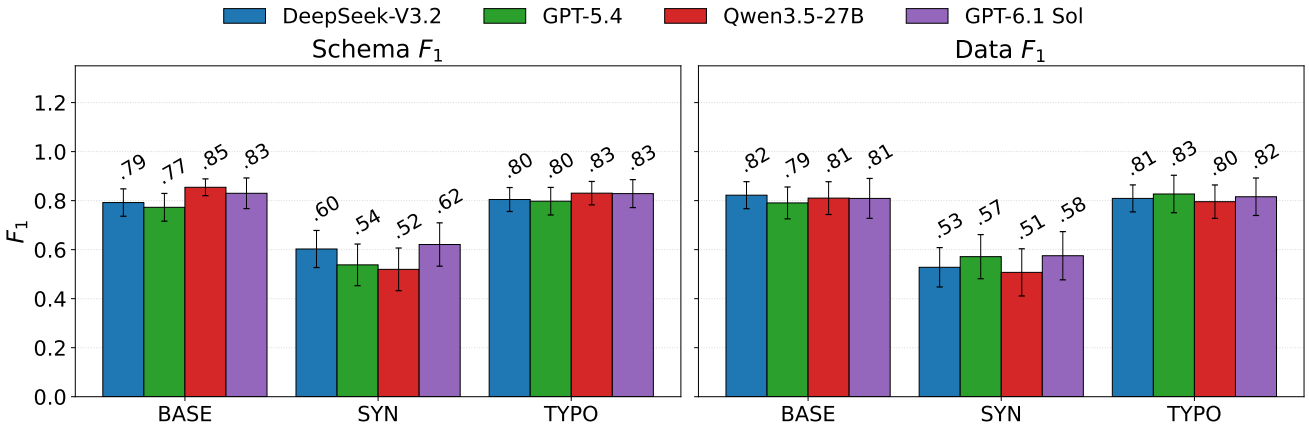

Figure 3. Schema and data F_1 under the lexical retrieval configuration, by input condition and LLM. Bars are per-question means; error bars are ± 1 SE across the questions of each cell. BASE and TYPO are level with one another, while SYN falls by 22 to 30 points. UNDER is scored behaviorally (cf. Section 6.3) and therefore not shown.

DeepSeek's SYN triple F_1 rises from 0.578 to 0.661 and its path coherence from 0.641 to 0.762.

Consistent with this reading, the mechanism is neutral where nothing is broken. On BASE and TYPO the differences between the two configurations are small for three models, and for DeepSeek the lexical variant is ahead on BASE by 0.06; for GPT-6.1 Sol the semantic variant is ahead by 0.10 on BASE and 0.05 on TYPO. At inference time the embedding calls add about two seconds of tool execution per run (4.2 s against 2.2 s) within a mean run time of 84 s that is the same for both configurations; the index itself is a one-off pass per dataset. Semantic retrieval is thus a remedy for vocabulary mismatch specifically, not a general

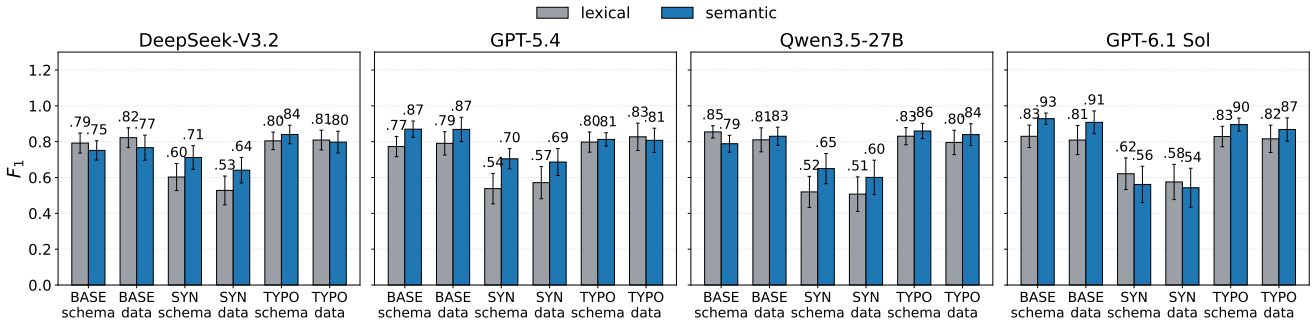

Figure 4. Lexical versus semantic retrieval across the three conditions with a defined answer, per LLM. Bars are per-question means; error bars are ± 1 SE across the questions of each cell. The semantic variant lifts SYN for three of the four models and differs little from the lexical one on BASE and TYPO.

improvement; Section 7.2 discusses the preparatory effort it requires.

The residual gap: Semantic retrieval does not restore BASE-level performance on average. Three factors account for the remaining gap, and the rest of this section treats them in turn: the rank of the target schema element in the embedding index, the naming of the entity in the question, and the reaction of the backbone to candidates that approximate the target. The first two are properties of the question and the schema and apply to every backbone; the third separates GPT-6.1 Sol from the other three.

Three groups of SYN questions: Two properties of the question and the schema determine whether semantic retrieval restores a question. The first is the rank of the target. Each SYN question replaces one or more words of its BASE counterpart, and since BASE uses the schema vocabulary verbatim, most of the replaced words name a schema element that the gold query requires. For each such replacement we embed the new wording with the tool's embedding model and read off the rank of that element among the elements of its type in the tool's own index; a question with several replacements is characterized by the rank of its lowest-ranking required element, because one element that the agent cannot find is enough to break the query. The tool returns the five nearest elements by default; the agents raised that number in 37% of their semantic search calls under SYN (against 22% under BASE) but never beyond 20 candidates in any of the 1,796 calls of the SYN runs, so a target beyond rank 50 was never within reach of any run. The second property is whether the question refers to the entity it filters on by a name the data does not contain. These two properties divide the 17 SYN questions into three groups. A *displaced target* is a question whose lowest-ranking required element lies beyond rank 50 (SYN13, SYN20). A *renamed entity* is a question that names its entity differently from the data (SYN16, SYN21). A *reachable target* is every other question (the remaining 13). Figure 5 shows the answer quality of the three groups.

Reachable targets: For the 13 reachable targets, semantic retrieval closes the gap for three backbones. Their SYN data F_1 reaches the BASE level for GPT-5.4 (0.80 vs. 0.83), DeepSeek (0.71 vs. 0.70), and Qwen (0.76 vs. 0.78), while lexical retrieval leaves the same questions at 0.51. Over all four backbones the group reaches 0.74 against 0.80; the difference is the GPT-6.1 Sol case treated below.

Displaced targets: The lowest-ranking required element is at rank 23 or better for 15 questions and at rank 90 and 722 for the two displaced targets, so any cut between rank 24 and rank 100 produces the same groups. In SYN13 the wording *oil field* has to find the class `Deposit` (rank 90) and *managing company* the property `activeDepositOperator` (rank 101); in SYN20 the wording *oil company* has to find that same property. For these two questions semantic retrieval is worse than lexical matching (0.25 vs. 0.78). The embedding of *oil company* ranks among the roughly 1,800 property elements of the NPD schema behind dozens of siblings of the target, such as `DepositOperator`, `operatorForDeposit`, and the licensee and facility operators, whereas a substring search for a guessed stem such as *operator* still reaches it. Panel (b) of Figure 5 shows the loss following the rank. The linguistic kind of the substitution recorded in the corpus (cf. Section 5.2) does not separate the questions: circumlocutions reach their target in SYN06, SYN08, SYN09, and SYN12 and miss it in SYN13 and SYN20, and the plain synonym *oil field* for `Deposit` is at rank 90 while *transit route* for `Line` is at rank 2.

Renamed entities: SYN16 asks for the fields operated by *Equinor*, which the data, a snapshot of the Norwegian petroleum registry, record under the company's former name *Statoil*, carried until 2018; SYN21 asks for the fields of *Aker BP*, which the data record under its two predecessor companies. For these two questions the failure is not one of schema retrieval. Their required classes and properties are at ranks 8 to 16 and are retrieved. What the schema cannot supply is the literal for the `FILTER`: a user who knows the company by its current name has asked a legitimate question, and the schema offers nothing to translate the name, so whether a run succeeds depends on world knowledge. GPT-5.4 and DeepSeek resolve *Equinor* to *Statoil* in every lexical run. *Aker BP* is resolved fully by GPT-5.4 in 5 of 12 runs and by none of the other three backbones.

Backbone reaction to approximate candidates: GPT-6.1 Sol reaches 0.68 against 0.88 on the reachable targets. Figure 6 and Table 3 show that the difference lies in the hand-back behavior. On BASE and TYPO the hand-back rate of GPT-6.1 Sol equals that of the other backbones. Under lexical retrieval on SYN, all four backbones hand back in 22 to 37% of the runs, because a lexical miss returns no candidate and every backbone recognizes an empty result.

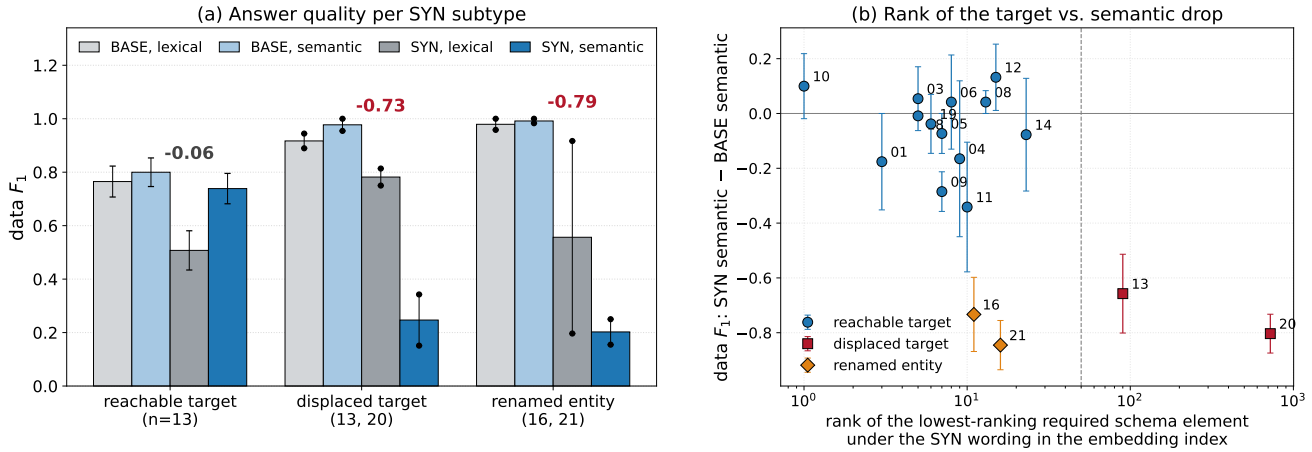

Figure 5. The three groups of SYN questions: reachable targets, displaced targets, and renamed entities. (a) Data F_1 per group for the four cells BASE/SYN $\times$ lexical/semantic; bars are per-question means over the four backbones, error bars ± 1 SE across questions, dots mark the single questions of the two small groups; the annotation is SYN semantic minus BASE semantic. (b) Per question, the rank of its lowest-ranking required schema element under the SYN wording in the tool's embedding index against that difference, error bars ± 1 SE across backbones; the dashed line is the rank cut of 50.

Semantic retrieval returns the nearest neighbors instead of an empty result, and all four backbones receive candidates outside the gold schema in 62 to 78% of their first SYN retrievals. The other three backbones generate a query from these candidates; their hand-back rate falls to 7 to 19%, and the affected runs end as partial answers with a mean data F_1 of 0.45 to 0.48. GPT-6.1 Sol hands back in 44% of the runs whose first retrieval missed a gold element, and its eight partial answers have a mean data F_1 of 0.67. The difference is concentrated in two questions. In SYN04 the generation agent requests confirmation that `Stop` denotes *platform* and `Line` denotes *transit route*; in SYN11 it requests the unit of `extractedCrude`, which the question does not state (*more than 50000000 in oil*; the data record millions of cubic metres, and GPT-5.4 inferred the scale from sampled values). The hand-back channel returns schema triples, not confirmations or units, so these runs end without a query. GPT-6.1 Sol therefore replaces approximate answers by requests that the pipeline cannot route, which is the behavior that Section 6.3 measures directly.

Table 3. How the 102 SYN runs per cell end, by backbone and retrieval configuration: fully correct (data $F_1 \geq 0.9$), partially correct ($0 < F_1 < 0.9$), wrong ($F_1 = 0$), or handed back to retrieval three times without a query. The two right-hand columns give the share of runs handed back on BASE and TYPO.

		SYN runs				handed back	
LLM	**Cfg**	**correct**	**partial**	**wrong**	**h. back**	**BASE**	**TYPO**
GPT-6.1 Sol	lex	55	8	7	32	13%	12%
	sem	50	8	13	31	3%	6%
GPT-5.4	lex	52	10	2	38	11%	10%
	sem	61	19	7	15	1%	4%
DeepSeek-V3.2	lex	42	29	9	22	3%	4%
	sem	51	32	12	7	6%	2%
Qwen3.5-27B	lex	45	10	10	37	2%	6%
	sem	54	16	13	19	3%	1%

All in all, RQ2 is answered as follows: embedding-based matching recovers the loss where the target element is within the search budget and the entity is named as in the data, which holds for 13 of the 17 questions. It does not recover the two questions whose target ranks beyond the budget or the two whose entity the data records under another name. For the strongest backbone the recovery is further limited by its own behavior: it requests confirmation where the other backbones generate a query from approximate candidates.

6.3 *Underspecified Questions (RQ3)*

RQ3 asks whether the agents recognize a question that cannot be answered without further input. An UNDER question does not determine a single SPARQL query: which query is correct depends on a decision the user has not made (cf. Section 5.2). Retrieval and result metrics therefore measure nothing for these questions, and the evaluation is behavioral instead: faced with a question it cannot answer without a decision of its own, the system should say so rather than commit to a reading of its own choosing.

We measure this in the two configurations of Section 3.2. The *default* configuration, which produced every other result in this paper, gives the agents no instruction concerning underspecification and is the behavior a deployed system shows unless its designer has anticipated the problem. The *instructed* configuration adds the one-sentence rule quoted in Section 3.2, and the BASE condition under the instructed prompt serves as the control for false alarms.

Table 4 reports the result. In the default configuration the outcome is categorical. In none of the 240 runs did any backbone report an underspecified question as such. Nine runs of Qwen3.5-27B under lexical retrieval did end without a query, but not for that reason. All nine fall on two questions, "Find large departments" and "Find significant emissions", and the reasons the agent recorded are that the schema contains no department class and no emission-related class. Both classes exist under the schema vocabulary, as `Division` and `ElementaryFlow`. The lexical search did not find them, and the agent reported a missing schema rather than an underspecified question. Under semantic retrieval the classes are found and all six runs of each question answer. These nine stops are retrieval failures of

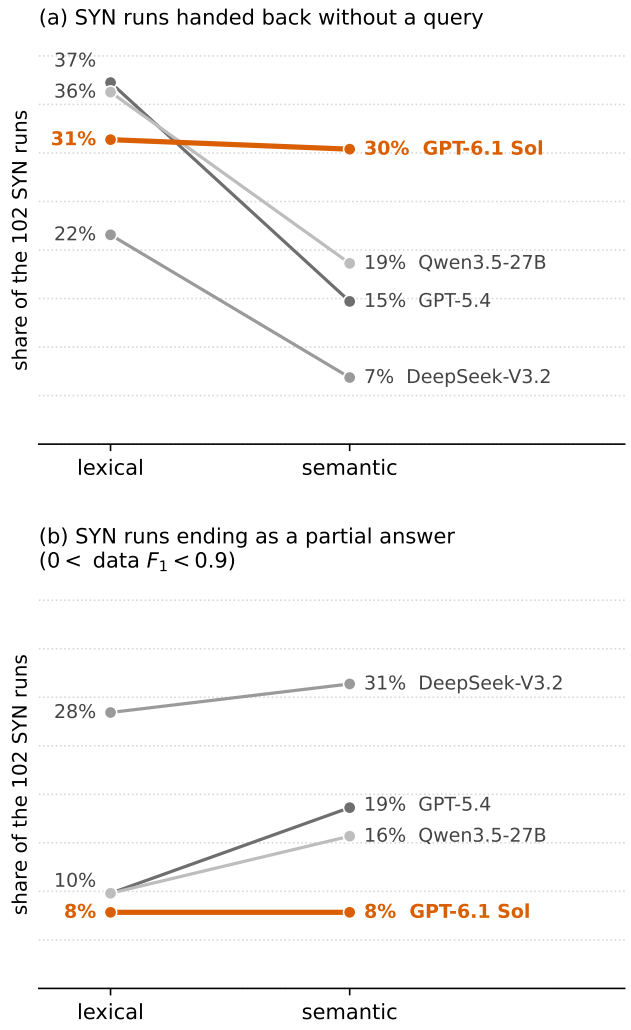

Figure 6. Effect of the retrieval configuration on the SYN runs, per backbone. (a) Share of runs handed back to retrieval three times without a query. (b) Share of runs ending as a partial answer. For three backbones the change from lexical to semantic retrieval converts hand-backs into partial answers; for GPT-6.1 Sol the hand-back share remains unchanged. Counts are given in Table 3.

the kind Section 6.1 describes, mistaken by the system for unanswerability, and Table 4 counts them as such.

The instructed columns of Table 4 show that the instruction is taken up, but very unevenly. GPT-6.1 Sol stops in every run, GPT-5.4 in about two thirds of its runs, DeepSeek-V3.2 in a third, Qwen3.5-27B almost never, and the retrieval configuration makes little difference. The control column shows the other half of the result. Not one BASE run was reported as underspecified, and the BASE runs that answered kept their data F_1 within three points of the uninstructed value for GPT-5.4, DeepSeek-V3.2, and Qwen3.5-27B and above it for GPT-6.1 Sol. Behind the totals lies a clear pattern by question. GPT-6.1 Sol stops on all five questions in every run. This is the same backbone that, under semantic retrieval on SYN, requested a confirmation or a unit instead of generating a query (Section 6.2); both measurements show the same disposition toward questions it cannot resolve from the schema alone. GPT-5.4 stops on four of the five questions in nearly every run. DeepSeek-V3.2 stops on three of them about half the time and never on "Show top products". Qwen3.5-27B stops only occasionally on two of them. The fifth question, "Get recent production data", is recognized in a single run across the latter three backbones: every other run silently fixed a year threshold. Where the agents stop, the reason they return names the missing criterion ("top by sales, rating, popularity, or price?", "what price threshold defines affordable?", "what date range should count as recent?"), although DeepSeek-V3.2 sometimes asks about the domain instead ("university or company departments?"). Qwen3.5-27B's lexical missing-schema stops persist at their default level under instruction: the instruction did not turn retrieval failures into recognitions.

The traces of the default runs show what the pipeline recorded while it answered. In every run that did not stop for a missing schema, the searches succeeded, since the schema for "products" is easy to find; the query executed, since it was well formed; the results were non-empty, since some reading of "top" always returns rows; and the verification checklist of Section 3.5 passed the query, since it checks the query against the schema and the data and no step checks the question against the intent. Under SYN, by contrast, the same instruments record a doubled search-failure rate, about two thirds more searches, and a retry rate of 38.2% for GPT-5.4 (cf. Section 6.1). Section 7.2 discusses the implications of this contrast.

All in all, RQ3 is answered in the negative for the default configuration: in none of the 240 runs did any backbone report an underspecified question, and every signal the pipeline records passed. Under instruction, recognition depends on the backbone, from every run for GPT-6.1 Sol and two thirds of the runs for GPT-5.4 to almost none for Qwen3.5-27B, with no false alarms on faithfully phrased questions.

7 Discussion

The results of Section 6 place the four input conditions into three classes, according to which part of the system has to handle them:

Absorbed by the model (TYPO). Corrected in context before the tool layer is reached. It requires nothing of the retrieval interface.

Partly repairable by retrieval (SYN). Reduces result quality under lexical matching; substantially, though not fully, recovered by embedding-based matching, at about two seconds of added tool time per run. The recovery requires a backbone that accepts the nearest neighbor as the match; GPT-6.1 Sol requests confirmation instead and recovers nothing.

Invisible to the system's signals (UNDER). Not a retrieval failure and not flagged by any execution or retrieval signal, so by default the system answers with unwarranted confidence. Detection is possible only as a check on the question itself, which the two strongest backbones perform once instructed, the strongest in every run, and the weaker ones largely do not. What sets UNDER apart is that it is detectable from the

Table 4. Recognition on the UNDER condition. *Default*: runs ending with the question reported as underspecified, and runs ending because the agent reported a missing schema element. *Instructed*: runs ending with the question reported as underspecified, and the share of BASE runs under the same prompt that were wrongly reported as underspecified (false alarms). Each UNDER cell covers five questions × six runs; the false-alarm control was run with semantic retrieval (17 BASE questions × three runs).

		default		instructed	
LLM	**retrieval**	**underspecified**	**missing schema**	**underspecified**	**false alarms (BASE)**
DeepSeek-V3.2	lexical	0/30	0/30	11/30	–
	semantic	0/30	0/30	9/30	0/51
GPT-5.4	lexical	0/30	0/30	20/30	–
	semantic	0/30	0/30	22/30	0/51
GPT-6.1 Sol	lexical	0/30	0/30	30/30	–
	semantic	0/30	0/30	30/30	0/51
Qwen3.5-27B	lexical	0/30	9/30	3/30	–
	semantic	0/30	0/30	0/30	0/51

question alone, before any tool call, whereas the ambiguities of the paired conditions surface only while the vocabulary or the instances are searched, and that only the user can settle it.

The rest of this section discusses what these classes imply for benchmark design, for system architecture, and for the limits of the study.

7.1 Implications for Benchmark Design

Our results bear on what a benchmark must contain if its scores are to predict how a system behaves in enterprise deployment. Current KGQA benchmarks were built for a different purpose, and three of their properties now look like liabilities rather than simplifications.

Lexical distance: Lexical distance, the gap between a question's wording and the schema vocabulary, has to become a reported variable. A benchmark whose questions predominantly reuse the vocabulary of the underlying schema measures a system under its most favorable condition, and its headline score cannot show how much of a result depends on that condition (cf. Section 1). In our paired sample the difference is 22 to 30 points of data F_1 under lexical retrieval, for the same questions and the same ground truth, and in enterprise KGQA the divergent regime is the normal case. A benchmark could satisfy the audit criteria of Zhang et al.[24] completely, with clean annotations and tolerant scoring, and still overstate real-world performance by containing only vocabulary-faithful questions.

What is required is that lexical distance be recorded, not that it be manipulated. Each existing question can be annotated with its distance to the schema elements its gold query targets, for instance as the embedding similarity between the question and those labels. Reporting accuracy against that variable rather than averaged over it already exposes whether a score depends on the faithful end of the distribution. Our sample also suggests that distance alone does not explain the outcome. The two SYN questions that semantic retrieval could not recover were no farther from their labels in embedding space than several it recovered, but their targets sat among dozens of near neighbours in a large schema (cf. Section 6.2). With two such questions this is an observation rather than a result, but it argues for recording, alongside the distance, how crowded the target's neighbourhood is, for instance as the target's rank among all schema elements under the question's wording. The paired design used here goes further, holding intent and ground truth fixed so that the gap is attributable to wording alone, a requirement for measuring the effect rather than merely detecting it. For benchmark construction the annotation is the more practical requirement, since it applies to corpora that already exist.

Ambiguity: Ambiguity has to be ruled out per question or scored for. A benchmark that scores an ambiguous question against a single gold query penalizes systems for choosing a reading its authors did not anticipate, and rewards the one that happens to share the authors' reading. There are three ways to avoid this, and a benchmark has to commit to one of them for every question. The first is to make sure that the question is unambiguous, so that exactly one reading is faithful to it. This is the cleaner option, but it is hard to guarantee: a reading the authors did not think of is, by definition, one they cannot rule out, and our own corpus shows how easily a criterion goes unstated. The second is to accept the ambiguity and to carry every faithful reading into the ground truth, so that a system is scored on whether its answer is correct under some reading rather than under the authors' reading. Our adaptive ground truth (cf. Section 4.1) takes this route, enumerating the admissible formulations per question and separating required from acceptable result columns. The third is to let the benchmark supply the missing information interactively: the question stays ambiguous, a clarification dialogue settles the open decision, and a single gold query holds for the question together with its dialogue. PRACTIQ[30] builds such dialogues for Text2SQL; what a protocol of this kind has to score is taken up in Section 7.3. The specific construction matters less than the requirement it meets. A benchmark must not do what most benchmarks do: leave the ambiguity in place and score against one reading.

Recognition: Recognition, whether a system declines to answer when it should, has to be measured alongside answer quality. The recognition result of Section 6.3 points to an omission that the first two of these options leave untouched. Text2SQL has begun to score abstention and clarification[27;30], but KGQA benchmarks ask whether a system answers correctly, not whether it declines to answer

when it should. A benchmark measuring only answer quality would record our agents' default willingness to answer every underspecified question as a moderate score rather than as the systematic failure to abstain that it is. We therefore argue that benchmarks for this setting should contain questions that *cannot* be answered from the data, scored on whether the system recognizes them as such. Such questions are straightforward to construct under control, since underspecification requires only omitting a criterion, and they measure a property that matters more in deployment than a few points of F_1: whether a user can tell the difference between an answer and a guess.

7.2 Implications for System Architecture

For practitioners the results carry a more immediate lesson. Granting the agent more attempts does not address retrieval failure: under vocabulary mismatch the agents already search far more than on faithful phrasings, and the additional searches fail at twice the rate (cf. Section 6.1). The matching mechanism, not the iteration budget, is the binding constraint.

Which mechanism to choose is a trade-off, and the results locate where it falls. Embedding-based matching recovers a large part of the SYN gap and reduces search effort while doing so, but it requires a preparatory pass over the data per dataset, a stored index to maintain as the sources evolve, and a dependency on an embedding model. Lexical substring matching requires none of this, can be pointed at a new dataset immediately, and is not weaker where the vocabulary is not in question (cf. Section 6.2). A deployment whose users are the schema's own authors, or one that reaches its vocabulary through a controlled interface, can therefore run on substring matching. The preparatory effort yields robustness to vocabulary mismatch specifically, so it is justified where that mismatch is expected and hard to justify where it is not.

For deployment, the recognition result is more relevant than the retrieval result. The traces of Section 6.3 show why. Every mechanism that might have caught an underspecified question is activated by a *failure signal*: the retrieval agent searches again when a search returns nothing, the generation agent revises when a query fails or returns no rows, and the verification step objects when the results do not fit the question. This separates two failure modes that look alike from the outside but differ in their diagnosability, and it reproduces for SPARQL what Wang et al.[27] report for Text2SQL. Vocabulary mismatch produces observable failure signals: searches come back empty, the agent retries, and the failure is visible to the system and to anyone monitoring it, even where it cannot be repaired. Underspecification produces none: every step succeeds under whichever reading the agent has adopted, and the system reports nothing unusual. The nine default stops under lexical retrieval make the same point from the other side: the only occasions on which the default system declined to answer an underspecified question were occasions on which retrieval had failed.

The user therefore receives an unqualified answer to a question the system decided for them, with no indication that a decision was made at all. This *silent failure* may be more consequential in deployment than a visible one, because it provides no external cue for verification. The concern extends beyond underspecification: wherever the system resolves an ambiguity by guessing rather than asking, the guess is invisible in the output, and the paired conditions contain such cases too (cf. Section 4.1). What distinguishes UNDER is that there no reading is faithful to the question, so the guess does not select among admissible answers but substitutes for a decision the question left open.

Detection is therefore the design problem, and it must precede clarification. A system cannot ask the user about an ambiguity it has not registered, and the signals such systems already collect do not register this one. Since retrieval succeeds, retrieval metrics cannot serve as a confidence proxy. If anything they are actively misleading here, reporting a clean run for the case that most warrants doubt. Recognizing underspecification requires reasoning about what a question leaves unstated, which is a check on the question rather than on the schema, and the default configuration never asks for that check. The instructed configuration adds it as a single prompt rule, and its result (Table 4) settles what the default runs leave open: the check is not beyond the architecture but must be asked for, and its ceiling is set by the backbone. The silent failure is thus a property of the default configuration rather than of the architecture. It is removed by instruction where the backbone can perform the check, it persists where the backbone cannot, and the over-caution such an instruction might have introduced did not occur.

7.3 Limitations and Future Work

These findings are subject to four limitations. The first concerns the UNDER condition, which is based on five questions against seventeen for each of the paired conditions. Two considerations bound what may be read into it. The recognition rate is a coarse binary criterion and captures only whether the system declared the question underspecified. A system might in principle signal its uncertainty in other ways, for instance by qualifying the answer, and our instrumentation would not record this. And with five questions no recognition rate can be estimated precisely. What the small sample does support is the categorical part of the result: no recognition at all by default, and a wide spread between backbones once instructed. A corpus of underspecified questions large enough to estimate recognition rates, and instrumentation for softer forms of hedging, would be needed to go further.

Second, the query corpus is curated rather than collected from real users. The substitutions in SYN are ordinary rather than adversarial, following the construction principle of Spider-Syn[5], but only a study with authentic user input can establish how the four conditions are distributed in practice. Third, we treat the OBDA backend as a black box: inefficiently modeled R2RML/RML mappings can yield SPARQL-to-SQL rewritings whose execution time grows prohibitively with source size, so a systematic study of mapping design and rewriting cost is a prerequisite for production-scale deployment. Fourth, each phase is limited to three iterations. Under lexical retrieval 39% of the SYN runs and 8% of the BASE runs exhaust the generation budget, and these are the runs with the lowest scores (mean data F_1 of 0.17 and 0.34). The finding that restarting retrieval seldom

rescues a run therefore holds for the budget of three iterations used here.

The most consequential piece of future work is a benchmark that meets the three requirements of Section 7.1 and, in addition, publishes the mapping layer alongside the graph, so that systems operating over mappings can be evaluated at all. Each requirement is modest on its own, and existing resources each meet one of them: PRACTIQ[30] scores clarification and unanswerability, Spider 2.0[16] covers enterprise workflows over large schemas, and Bolleman et al.[49] collect over 1,000 human-written question–query pairs over federated graphs. None combines them. Our corpus of 56 question formulations (cf. Section 5.2) is too small to serve this purpose, and the datasets we could assemble were limited to the five that ship the complete triple of data, ontology, and mappings. Building such a resource is a community undertaking rather than a by-product of a single study, and without it the effects reported here cannot be tracked across systems.

That benchmark faces one problem we did not have to solve. Our agents answer or decline, and both are scoreable against a fixed corpus. A system that *asks* instead (cf. Section 5.2) changes the question it ends up answering, which single-turn question–answer pairs cannot express. PRACTIQ[30] constructs such clarification dialogues for Text2SQL; no comparable resource exists for KGQA over mappings. A protocol for human-in-the-loop Text2Query would have to score whether the system asks when it should and stays silent when it should not, whether its question identifies the decision actually left open, and how many turns it needs, whether with simulated users, recorded clarification dialogues, or a rubric over the questions asked.

In addition, three narrower extensions follow directly from the study. The instructed configuration of RQ3 tests a single instruction. Whether richer prompting, a clarification turn with the user, or fine-tuning shifts recognition further is open. It is worth settling before treating recognition as a property of the model rather than of the prompt. Folding `rdfs:comment`, `skos:definition`, alternative labels, and richer axioms (domain and range, disjointness) into the schema-extraction step is a low-cost way to reduce the displaced-target subtype of the residual SYN gap (cf. Section 6.2), since a definition such as "a deposit is an oil or gas field" would move the target up the ranking. The renamed-entity subtype instead calls for entity resolution against the data or an external source of world knowledge. Extending the study to federated questions that require joining concepts across sources would test whether the retrieval/question distinction established here holds when the schema itself spans providers.

8 Conclusions

Text2SQL research established that questions phrased without the schema's own vocabulary degrade query generation substantially. We asked whether tool-using agents, which can search and revise rather than answer in one pass, absorb this effect on their own, and we asked it for enterprise KGQA over R2RML/RML mappings, where the vocabulary is proprietary and no graph needs to be materialized. Answering it required a two-agent system whose retrieval stage can be swapped between lexical and embedding-based matching while everything else is held fixed, and an evaluation framework that scores ambiguous questions against every faithful reading and attributes a failure to the stage of the pipeline that caused it.

Tool access alone does not confer robustness. Across four LLM backbones the agents respond to vocabulary mismatch by searching more and failing more often, and result quality still falls by 22 to 30 points of data F_1. The loss is reduced by changing the matching mechanism, and the reduction is confined to the source of the problem: embedding-based retrieval recovers the loss completely where the schema element is within reach of the index (13 of 17 questions), not at all where the target ranks beyond the agents' search budget or the question names an entity under a name the data does not carry, and it is neutral on faithfully phrased and misspelled questions.

The most consequential result concerns what the agents do not report. Presented with questions whose missing decision criterion cannot be recovered from the schema, no backbone said so in a single default run. Once instructed, the strongest did so in every run and the second strongest in two thirds of its runs, without a false alarm on well-posed questions, the weakest almost never. A retrieval failure produces signals the system detects even when it cannot repair it. This failure produces none, for the user as much as for the system.

Wording effects thus sort into three classes: absorbed by the model, repairable by retrieval, and left unreported. Distinguishing them is not a detail of system tuning but a condition for measuring progress in this setting at all. A benchmark that only rewards answering cannot distinguish a system that knows the answer from one that has quietly invented the question, and the benchmark revisions of Section 7.1 are meant to make this possible.

Acknowledgements

During the preparation of this work, the authors used GPT-5.5 and Claude (Opus 4.6/4.7, Fable 5.1) to check grammar and spelling, to draft revisions of individual passages that the authors rewrote and approved, and to write and improve the code, and an image-generation model for the two mascot illustrations in Figure 1. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

Author contributions

Conceptualization, J.D.; methodology, J.D.; software, J.D.; validation, J.D.; formal analysis, J.D.; investigation, J.D.; data curation, J.D.; writing—original draft preparation, J.D.; writing—review and editing, J.D., T.M. and A.P.; visualization, J.D.; supervision, T.M. and A.P.; project administration, A.P. All authors have read and agreed to the published version of the manuscript.

Statements and declarations

Ethical considerations

Not applicable. The study involves no human participants, animals, or personal data.

Consent to participate

Not applicable.

Consent for publication

Not applicable.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

Data availability

The source code for the pipeline, the evaluation framework, and the R2RML-to-semantic-mapping transformation, together with the question corpus, the adaptive ground truth, the mappings and ontologies, the evaluated results of every run, and the scripts that produce every table and figure, are publicly available at https://github.com/JakobDch/vkgqa-vocabulary-mismatch. The raw traces of all runs and the database dumps of the five datasets in the state used for the experiments are distributed as archives attached to the repository's release for this submission. LUBM, GTFS, NPD, and BSBM are in addition available from their respective original sources; the LCA data was provided by the ORIENTING project.

Abbreviations

GT	Ground truth
KGQA	Knowledge Graph Question Answering
LLM	Large Language Model
NL	Natural language
OBDA	Ontology-Based Data Access
RML	RDF Mapping Language
R2RML	RDB to RDF Mapping Language
SC	Semantic concept
VKGQA	Virtual Knowledge Graph Question Answering

References

1. Guo J, Zhan Z, Gao Y et al. Towards complex Text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*. pp. 4524–4535. DOI: 10.18653/v1/P19-1444.
2. Wang B, Shin R, Liu X et al. RAT-SQL: Relation-aware schema encoding and linking for Text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*. pp. 7567–7578. DOI: 10.18653/v1/2020.acl-main.677.
3. Sakor A, Singh K, Patel A et al. Falcon 2.0: An entity and relation linking tool over Wikidata. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020)*. pp. 3141–3148. DOI: 10.1145/3340531.3412777.
4. Dubey M, Banerjee D, Chaudhuri D et al. EARL: Joint entity and relation linking for question answering over knowledge graphs. In *Proceedings of the 17th International Semantic Web Conference (ISWC 2018)*. pp. 108–126. DOI:10.1007/978-3-030-00671-6_7.
5. Gan Y, Chen X, Huang Q et al. Towards robustness of Text-to-SQL models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 2505–2515. DOI:10.18653/v1/2021.acl-long.195.
6. Steinmetz N and Sattler KU. What is in the KGQA benchmark datasets? Survey on challenges in datasets for question answering on knowledge graphs. *Journal on Data Semantics* 2021; 10(3–4): 241–265. DOI:10.1007/s13740-021-00128-9.
7. Talaei S, Pourreza M, Chang YC et al. CHESS: Contextual harnessing for efficient SQL synthesis. In *ICML 2025 Workshop on Multi-Agent Systems in the Era of Foundation Models: Opportunities, Challenges and Futures*.
8. Pourreza M and Rafiei D. DIN-SQL: Decomposed in-context learning of Text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. pp. 36339–36348.
9. Gao D, Wang H, Li Y et al. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment* 2024; 17(5): 1132–1145. DOI: 10.14778/3641204.3641221.
10. Mavromatis C, Adeshina S, Ioannidis VN et al. BYOKG-RAG: Multi-strategy graph retrieval for knowledge graph question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 27881–27898. DOI:10.18653/v1/2025.emnlp-main.1417.
11. Lin C, Jiang Z, Zheng L et al. RJE: A retrieval-judgment-exploration framework for efficient knowledge graph question answering with LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 17277–17294. DOI:10.18653/v1/2025.emnlp-main.873.
12. Wang Z, Zheng Y, Cao Z et al. AutoLink: Autonomous schema exploration and expansion for scalable schema linking in text-to-SQL at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40. pp. 33809–33817. DOI: 10.1609/aaai.v40i40.40672.
13. Yu T, Zhang R, Yang K et al. 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 (EMNLP 2018)*. Brussels, Belgium: Association for Computational Linguistics, pp. 3911–3921. DOI:10.18653/v1/D18-1425.
14. Usbeck R, Yan X, Perevalov A et al. QALD-10 – the 10th challenge on question answering over linked data: Shifting from DBpedia to Wikidata as a KG for KGQA. *Semantic Web* 2024; 15(6): 2193–2207. DOI:10.3233/SW-233471.
15. Dubey M, Banerjee D, Abdelkawi A et al. LC-QuAD 2.0: A large dataset for complex question answering over Wikidata and DBpedia. In *The Semantic Web – ISWC 2019*, *Lecture Notes in Computer Science*, volume 11779. Springer, pp. 69–78. DOI:10.1007/978-3-030-30796-7_5.
16. Lei F, Chen J, Ye Y et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*.
17. Sequeda J, Allemang D and Jacob B. A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise SQL

databases. In *Proceedings of the 7th Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*. DOI:10.1145/3661304.3661901.

18. Hauff M, Molinas Comet L, Moosmann P et al. FAIRness in dataspaces: The role of semantics for data management. In *Proceedings of the 2nd International Workshop on Semantics in Dataspaces (SDS 2024), co-located with the 21st Extended Semantic Web Conference (ESWC 2024)*, *CEUR Workshop Proceedings*, volume 3705. CEUR-WS.org.
19. Li J, Hui B, Qu G et al. Can LLM already serve as a database interface? A BIg bench for large-scale database grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) Datasets and Benchmarks Track*. pp. 42330–42357.
20. Deshmukh RA, Collarana D, Gelhaar J et al. Challenges and opportunities for enabling the next generation of cross-domain dataspaces. In *Proceedings of the 2nd International Workshop on Semantics in Dataspaces (SDS 2024), co-located with the 21st Extended Semantic Web Conference (ESWC 2024)*, *CEUR Workshop Proceedings*, volume 3705. CEUR-WS.org.
21. Arenas-Guerrero J, Chaves-Fraga D, Toledo J et al. Morph-KGC: Scalable knowledge graph materialization with mapping partitions. *Semantic Web* 2024; 15(1): 1–20. DOI:10.3233/SW-223135.
22. Calvanese D, Cogrel B, Kalayci EG et al. OBDA with the Ontop framework. In *Proceedings of the 23rd Italian Symposium on Advanced Database Systems (SEBD 2015)*. pp. 296–303.
23. Deich J, Elbl F, Meisen T et al. Virtual knowledge graph question answering: Bridging the application gap for semantics in dataspaces, 2026. Accepted paper at the 4th International Workshop on Semantics in Dataspaces (SDS 2026), co-located with the 23rd Extended Semantic Web Conference (ESWC 2026). To appear in CEUR Workshop Proceedings.
24. Zhang L, Jiang Z, Chi H et al. Diagnosing and addressing pitfalls in KG-RAG datasets: Toward more reliable benchmarking. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025), Datasets and Benchmarks Track*.
25. Deng X, Awadallah AH, Meek C et al. Structure-grounded pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2021)*. Association for Computational Linguistics, pp. 1337–1350. DOI:10.18653/v1/2021.naacl-main.105.
26. Chang S, Wang J, Dong M et al. Dr.Spider: A diagnostic evaluation benchmark towards text-to-SQL robustness. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.
27. Wang B, Gao Y, Li Z et al. Know what I don't know: Handling ambiguous and unknown questions for Text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto, Canada: Association for Computational Linguistics, pp. 5701–5714. DOI:10.18653/v1/2023.findings-acl.352.
28. Wretblad N, Riseby F, Biswas R et al. 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 (Volume 2: Short Papers)*. Bangkok, Thailand: Association for Computational Linguistics, pp. 356–369. DOI:10.18653/v1/2024.acl-short.34. URL https://aclanthology.org/2024.acl-short.34/.
29. Gomm D, Wolff C and Hulsebos M. Are we asking the right questions? On ambiguity in natural language queries for tabular data analysis. In *EurIPS 2025 Workshop: AI for Tabular Data*. URL https://openreview.net/forum?id=Mm1CGvmm9Z.
30. Dong M, Ashok Kumar N, Hu Y et al. PRACTIQ: A practical conversational text-to-SQL dataset with ambiguous and unanswerable queries. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 255–273. DOI:10.18653/v1/2025.naacl-long.13.
31. Liu G, Tan Y, Zhong R et al. Solid-SQL: Enhanced schema-linking based in-context learning for robust text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*. Association for Computational Linguistics, pp. 9793–9803.
32. Perevalov A and Both A. Text-to-SPARQL goes beyond English: Multilingual question answering over knowledge graphs through human-inspired reasoning. In *Proceedings of the First International TEXT2SPARQL Challenge 2025, co-located with Text2KG at ESWC25*, *CEUR Workshop Proceedings*, volume 4094. Portorož, Slovenia: CEUR-WS.org, pp. 77–93. URL https://ceur-ws.org/Vol-4094/paper6.pdf.
33. Usbeck R, Gusmita RH, Ngonga Ngomo AC et al. 9th challenge on question answering over linked data (QALD-9). In *Joint Proceedings of SemDeep-4 and NLIWoD-4 and the 9th QALD Challenge co-located with ISWC 2018*, *CEUR Workshop Proceedings*, volume 2241. CEUR-WS.org, pp. 58–64.
34. Lee J and Shin H. SPARKLE: Enhancing SPARQL generation with direct KG integration in decoding. *Expert Systems with Applications* 2025; 289: 128263. DOI:10.1016/j.eswa.2025.128263.
35. Rony MRAH, Kumar U, Teucher R et al. SGPT: A generative approach for SPARQL query generation from natural language questions. *IEEE Access* 2022; 10: 70712–70723. DOI:10.1109/ACCESS.2022.3188714.
36. D'Abramo J, Zugarini A and Torroni P. Investigating large language models for text-to-SPARQL generation. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing (KnowledgeNLP 2025)*. Albuquerque, New Mexico, USA: Association for Computational Linguistics, pp. 66–80. DOI:10.18653/v1/2025.knowledgenlp-1.5.
37. Pliukhin D, Radyush D, Kovriguina L et al. Improving subgraph extraction algorithms for one-shot SPARQL query generation with large language models. In *Joint Proceedings of the 1st Scholarly QALD Challenge and the 4th SemREC Challenge, co-located with ISWC 2023*, *CEUR Workshop Proceedings*, volume 3592. CEUR-WS.org. URL https://ceur-ws.org/Vol-3592/paper6.pdf.
38. Emonet V, Bolleman J, Duvaud S et al. LLM-based SPARQL query generation from natural language over federated knowledge graphs. In *Proceedings of the Special Session on Harmonising Generative AI and Semantic*

Web Technologies (HGAIS 2024), co-located with the 23rd International Semantic Web Conference (ISWC 2024), *CEUR Workshop Proceedings*, volume 3953. Baltimore, Maryland, USA: CEUR-WS.org. URL https://ceur-ws.org/Vol-3953/355.pdf.

39. Gómez SA and Fillottrani PR. Leveraging large language models for ontology-based data access: A preliminary analysis. In *XXX Congreso Argentino de Ciencias de la Computación (CACIC 2024)*. La Plata, Argentina. ISBN 978-950-34-2428-5, pp. 996–1005.
40. Askar M, Algergawy A, Soliman THA et al. Ontology based natural language queries transformation into SPARQL queries. *Baltic Journal of Modern Computing* 2020; 8(4): 719–731. DOI:10.22364/bjmc.2020.8.4.14.
41. Allemang D and Sequeda J. Increasing the accuracy of LLM question-answering systems with ontologies. In *The Semantic Web – ISWC 2024*, *Lecture Notes in Computer Science*, volume 15233. Cham: Springer, pp. 324–339. DOI: 10.1007/978-3-031-77847-6_18.
42. Curry E. *Real-time Linked Dataspaces: Enabling Data Ecosystems for Intelligent Systems*. Cham: Springer International Publishing, 2020. DOI:10.1007/978-3-030-29665-0.
43. Guo Y, Pan Z and Heflin J. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics* 2005; 3(2–3): 158–182. DOI:10.1016/j.websem.2005.06.005.
44. Chaves-Fraga D, Priyatna F, Cimmino A et al. GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain. *Journal of Web Semantics* 2020; 65: 100596. DOI:10.1016/j.websem.2020.100596.
45. Lanti D, Rezk M, Xiao G et al. The NPD benchmark: Reality check for OBDA systems. In *Proceedings of the 18th International Conference on Extending Database Technology (EDBT 2015)*. OpenProceedings.org, pp. 617–628. DOI:10.5441/002/edbt.2015.62.
46. Bizer C and Schultz A. The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems* 2009; 5(2): 1–24. DOI:10.4018/jswis.2009040101.
47. Baumanns L, Deich J, Meisen T et al. Semantic modeling of life cycle assessment data using knowledge graphs, 2026. Accepted paper at the 4th International Workshop on Knowledge Graphs for Sustainability (KG4S 2026), co-located with the 23rd Extended Semantic Web Conference (ESWC 2026). To appear in CEUR Workshop Proceedings.
48. Sainz O, Campos JA, García-Ferrero I et al. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. pp. 10776–10787. DOI:10.18653/v1/2023.findings-emnlp.722.
49. Bolleman J, Emonet V, Altenhoff A et al. A large collection of bioinformatics question–query pairs over federated knowledge graphs: methodology and applications. *GigaScience* 2025; 14: giaf045. DOI:10.1093/gigascience/giaf045.

A Worked Example of the Result-Level Evaluation

This appendix walks through the running example shown in Figure 2 step by step.

Step 1 (admissible GT queries): The NL query is *"Find boreholes drilled before 2010 with a total drill depth greater than 3000 m."* As discussed in Section 4.1, `eno:totalDrillDepth` and `eno:finalVerticalDrillDepth` are both plausible readings of *total drill depth*, so we include two admissible GT queries, Q_1 using the former and Q_2 the latter. The wellbore identifier, depth, and start date are REQUIRED; the human-readable designation is ACCEPTABLE and wrapped in `OPTIONAL`, so that wellbores without a designation still appear in the solution.

Step 2 (admissible GT solutions): $G_q^{(1)} = \mathrm{sol}(Q_1)$ contains five tuples; the row `:w5` has no designation, but the `OPTIONAL` clause keeps it in the result with a `NULL` placeholder for that column. $G_q^{(2)} = \mathrm{sol}(Q_2)$ contains three tuples, since `finalVerticalDrillDepth` is missing for two wellbores.

Step 3 (column matching): The system result L_q contains three columns: an identifier, a label, and a depth. Pairing them by μ_Q (cf. Section 4.1), all three pair successfully against Q_1. Against Q_2 the depth columns do not: the system's depth values come from `totalDrillDepth` and do not co-occur with `finalVerticalDrillDepth` on the same wellbores, so the property restriction sets ω_{Q_2} to zero.

Step 4 (candidate selection): Schema recall is 1.00 for Q_1 (all REQUIRED concepts covered) and 0.50 for Q_2 (depth concept uncovered; its schema F_1 is 0.57 against 1.00 for Q_1). We select $Q^* = Q_1$ as the matching GT solution and compute data metrics on its paired projection: $P_{\text{data}} = 2/3$, $R_{\text{data}} = 1/2$, $F_{1,\text{data}} = 0.57$.

B Schema Extraction Procedure

This appendix specifies the extraction step of Section 3.3 at the level needed to reproduce it. Its inputs are the mapping document (R2RML or RML), the OWL ontology, and a dump of the source (an SQL dump or a folder of CSV files).

1. *Source sample.* The extractor parses the table definitions of the dump and reads up to 100 rows per table (for CSV sources every column is typed as text), then loads them into an SQLite stub with the column types the dump declares. No values are synthesized: templates, datatypes, and join conditions are instantiated with the source's own values, so the schema reflects what the mapping produces on real data.

2. *Materialization.* Morph-KGC evaluates the unmodified mapping over the stub. `rr:sqlQuery` views, `rr:joinCondition`, and `rr:parentTriplesMap` are handled by the engine; an object property materializes when the sample contains a matching key pair on both sides of the join.

3. *Schema derivation.* For every subject that carries an `rdf:type`, each of its predicates becomes a data property when the object is a literal, with the literal's datatype and `xsd:string` when it has none (which also covers language-tagged literals), and an object property when the object is an IRI with a type, with that type as the target class. Blank-node objects are recorded like literals. Entries are deduplicated per class on

(predicate, datatype) and (predicate, target class), which also collapses duplicate mappings of the same predicate.

4. *Ontology.* The ontology file is loaded as given; `owl:imports` are not followed. For every class with materialized subclasses, the extractor adds an entry with the union of the subclasses' properties and lifts object-property targets to the most general superclass that still has materialized subclasses.
5. *Output.* One Turtle file per class in the notation of Section 3.1, which the retrieval tools index.

Coverage: Because step 2 evaluates the mapping over a sample, a mapped property enters $\mathcal{S}$ when the sample instantiates it: a column that is empty in every sampled row, or a join without a partner in the sample, leaves no trace. Table A1 reports, per dataset, how many of the properties the mapping declares (the predicates of its predicate-object maps, excluding `rdf:type`) appear in $\mathcal{S}$. Every class and property that any of the 155 gold readings of the corpus requires is contained in $\mathcal{S}$, so the sample determines the size of the search space the agents face, not whether the questions can be answered.

Table A1. Share of the properties declared in each mapping that the extracted schema $\mathcal{S}$ contains.

Dataset	mapped properties	in $\mathcal{S}$	share
EDU	18	18	100%
TRN	56	44	79%
NRG	297	273	92%
BSBM	40	39	98%
LCA	20	15	75%
all	431	389	90%

C Text2SQL Reference Point

The pipeline used in this study answers questions over mappings rather than over a single relational schema, and Section 2.4 argues that this matters when one concept is distributed across structurally different provider tables. This appendix substantiates that argument, which motivates the semantic layer but is not itself a contribution of this paper.

We compare against CHESS[7] on the 17 SYN questions, with GPT-5.4 as the common backbone. CHESS is run in its released *Information Retriever* → *Candidate Generator* → *Unit Tester* configuration over SQLite snapshots of the same datasets, using the upstream prompts, parsers, and decoding parameters without modification. It exposes no retrieval phase, so only result-level metrics are comparable. Two data regimes are used. In the *heterogeneous* regime, each source table is duplicated under a synonymous name with renamed columns and the rows split evenly between the two, so that each concept lives in two structurally divergent tables, the situation that arises when several providers contribute the same concept. In the *consolidated* regime, each concept is materialized as a single canonical table, the regime in which Text2SQL benchmarks such as BIRD report strong results; we include it to upper-bound CHESS under conditions favorable to it.

Table A2. Schema and data F_1 over the 17 SYN questions (VKGQA: means over $17 \times 6 = 102$ traces; CHESS: one run per question).

Approach	N	Schema F_1	Data F_1
VKGQA semantic (GPT-5.4)	102	0.709	0.686 [1]
CHESS, heterogeneous (GPT-5.4)	17	0.724	0.543
CHESS, consolidated (GPT-5.4) [1]	17	**0.802**	**0.737**

[1] Upper bound obtained by re-executing the exact CHESS-produced SQL against a consolidated snapshot.

In the heterogeneous regime CHESS attains a competitive schema F_1, because each sub-table remains discoverable by its column names, but its data F_1 falls well below VKGQA: it selects one of the two sub-tables and queries it directly, whereas the correct answer requires unioning both. VKGQA handles this by construction, since both sub-tables share a mapping target in the ontology and the semantic layer presents them as one virtual class.

The third row isolates the cause. Re-executing the *exact* SQL that CHESS produced against a consolidated snapshot raises both metrics substantially, confirming that the penalty stems from the data split rather than from a failure to identify the relevant tables. With the split removed, CHESS outperforms VKGQA, recovering the picture familiar from BIRD-style benchmarks. The mapping layer therefore justifies its effort only where it adds information that the relational schema does not already contain, which is the condition under which federated settings are worth modeling semantically at all.