



Electronic Communications of the EASST
Volume 86 (1) Year 2026

deRSE26 - Selected Contributions of the 6th Conference for Research Software Engineering in Germany (Part 1)

Edited by Bernd Flemisch, René Caspart, Bernadette Fritzsch, Florian Goth, Jean-Noël Grad, Florian Thiery

Enhancing Accessibility to Semantic Search in LinkAhead using Large Language Models

Alexander Schlemmer, Daniel Hornung, Sara Kaviani

DOI: 10.14279/eceasst.v86.1.2778

License: This article is licensed under a CC-BY 4.0 License.

Electronic Communications of the EASST (https://eceasst.org).
Published by **Berlin Universities Publishing**
(https://www.berlin-universities-publishing.de/)

Enhancing Accessibility to Semantic Search in LinkAhead using Large Language Models

Alexander Schlemmer [1], **Daniel Hornung** [1], **Sara Kaviani** [1]

[1] a.schlemmer@indiscale.com
IndiScale GmbH, Göttingen, Germany

Abstract:

LinkAhead is an open-source research data management platform with a powerful semantic search engine. Queries are expressed in CQL (CaosDB Query Language), a domain-specific language that supports complex traversals of LinkAhead's graph-structured data model. Despite its expressive power, CQL syntax remains a barrier for non-technical users. We present LINDI, a translation pipeline that converts natural language queries into valid CQL without requiring users to learn the query language. The pipeline decomposes translation into a sequence of focused Large Language Model (LLM) calls, producing a Query Intermediate Representation (QIR) that is validated against the active data schema before query execution. The QIR is human-readable and directly editable, giving users a transparent checkpoint to inspect and correct before a query runs. We evaluate LINDI on 200 manually compiled queries across four datasets (three in English, one in German) from independent research domains. On the three English-language datasets, Llama 3.1 (8B) achieves consistent accuracy between 90.1% and 97.7%. GPT-OSS (20B) ranges from 91.5% to 93.2%. Overall accuracy across all four datasets is 88.0% and 85.0% for the two models respectively, and the pipeline's modular design allows the LLM component to be replaced independently. A single-step baseline that skips this decomposition reaches only 70.5%/72.5% overall accuracy on the same queries, indicating that decomposing translation into schema-aware, LLM-assisted steps with human review is a practical route to making LinkAhead's semantic search accessible without requiring users to learn CQL.

Keywords: Large Language Models, Natural Language Querying, Query Generation, CQL, Semantic Data Management

1 Introduction

LinkAhead [HSW] (formerly CaosDB [FSH+]) is an open-source semantic research data management system (RDMS). In contrast to data repositories and electronic lab notebooks, its main focus is on data management for actively evolving research data, especially during processing, analysis, and workflow execution. It includes functionality for workflow management that has been demonstrated in multiple applications [Fre24, SWF+19, LS26]. LinkAhead organizes research data as Records, individual data entities that instantiate a RecordType, a user-defined schema class describing which Properties a Record of that type may carry. For example, a Record

of type Sample might carry a Property `collection_date` holding a literal date value, and a Property `derivedFrom` referencing another Record. Properties can hold literal values (numbers, strings, dates) or reference other Records, so RecordTypes and their Properties together define a graph-structured, semantically typed data model rather than a fixed relational schema; these are exactly the Properties that queries filter on and traverse across Records, as described next. Fig. 1 shows an example instance of this model.

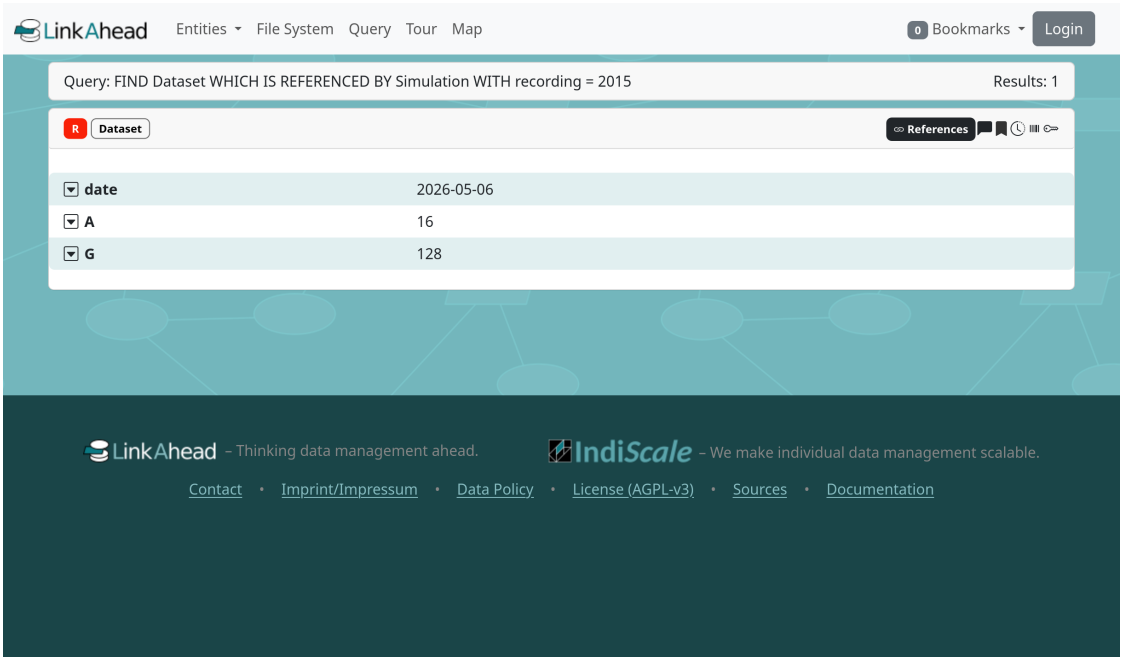


Figure 1: Screenshot of LinkAhead showing an example dataset. The example query that is described in Sec. 3.1 can be seen in the top panel.

Most of these use-cases rely on a semantic search engine that allows for specifically querying entities from the system possibly traversing linked entities. The main search engine used for this purpose is the integrated domain-specific language "CQL"[1] [HSW] that mimics SQL, but is specialized to LinkAhead's data model (see e.g. [FSH+] or the official documentation [Lin26c]). This search capability is also what makes LinkAhead valuable as a combined query-frontend when it is used to integrate multiple third-party RDMS via its crawler-framework [WSL+, SSTW25].

CQL (see also Sec. 3.1) resembles a formal query language and was designed with SQL in mind. Queries look like a simplified version of English, however there still is a fixed syntax and semantics that has to be taken into account while searching the RDMS. In our experience, researchers from natural sciences and related disciplines with a decent understanding of formal languages, like programming languages, tend to find CQL easy to learn. However, we have observed that many users from other disciplines find designing queries difficult, especially when

[1] CQL is the abbreviation for CaosDB Query Language. As CaosDB was renamed to LinkAhead, the language will also be renamed to LQL, LinkAhead Query Language, in the future.

the traversal of references is involved.

One approach to tackle this issue was to introduce query templates, which are customizable UI elements including placeholders, that are converted to pre-defined queries. More recently, as large language model technology found more widespread adoption [PG24], a dedicated translation component from natural language to CQL emerged as a promising application.

In this article we present LINDI, a translation system that facilitates access to semantic queries in LinkAhead using LLMs. The system is structured as a modular four-layer pipeline (Sec. 3.3) that combines focused LLM calls with schema-aware context narrowing (Layer 1), rule-based post-processing (Layers 2–3), and a human-in-the-loop review step (Layer 4), enabling reliable translation without dependence on large proprietary models.

2 Related work and Motivation

Translating natural language into structured database queries has been studied extensively under the name Text-to-SQL [DCZ22]. A consistent finding in this literature [YZY+18, YHY+24] is that single-step generation of the target query language is unreliable for queries involving complex joins, domain-specific syntax, or schema elements absent from pre-training data. This motivates our use of a simplified intermediate representation (QIR, see Sec. 3.2) rather than targeting CQL or SQT (see also Sec. 3.1) directly. Analogous work on natural language to SPARQL translation [MRO+25] addresses knowledge-graph querying, which is structurally closer to LinkAhead's graph-based data model; error analyses in that setting consistently identify schema misinterpretation as a leading failure mode, reinforcing our choice to include schema-aware generation and post-generation validation in the pipeline.

Beyond research systems, recent infrastructure frameworks also aim to connect LLMs to structured data systems. The Model Context Protocol (MCP) [Ant24] provides a mechanism for integrating external tools and data sources into LLM-based chat systems. It is a natural candidate for adding natural language access to custom databases. However, MCP-based integrations rely on the underlying LLM to generate queries directly, which does not meet our precision requirements for scientific data management. In principle, the translation component presented in this paper could be wrapped in an MCP server to provide reliable NL-to-SQT translation within MCP-compatible clients.

One of the main challenges of using LLMs in scientific data management is their tendency to hallucinate [JLF+23], i.e., to generate plausible but incorrect outputs. Because LinkAhead is primarily used in research environments, the system requires a high degree of precision and reliability. Current LLMs do not achieve sufficient accuracy to be trusted with unrestricted query generation without human oversight [BCP+26]. For this reason, LINDI includes a validation workflow that allows users to inspect and verify the generated query before execution.

Natural language queries also introduce an additional class of problems. Some queries may be ambiguous, underspecified, or impossible to map to the active data schema. Instead of attempting to infer unsupported semantics, the system explicitly detects such cases and returns descriptive feedback to the user. This prevents invalid or misleading queries from silently producing incorrect results (see Sec. 5.2).

In addition, dependence on large proprietary models introduces practical concerns related to

digital sovereignty [PC26], deployment flexibility, computational cost, and data privacy. We therefore aim to support smaller self-hosted models whenever possible. These requirements strongly influence the design of LINDI and motivate the modular translation pipeline described in the following section.

Taken together, LINDI's contribution is an applied instantiation of established patterns from the Text-to-SQL and NL-to-SPARQL literature (intermediate representations, schema-aware context narrowing, validation layers), adapted to LinkAhead's specific query language and graph-structured data model, rather than a new method in the NLP sense.

3 Methods

We evaluated several approaches for translating natural language queries into semantic queries for use with LinkAhead. A first attempt involved fine-tuning a language model based on GPT-2 [RWC+19] on a corpus of natural-language and CQL query pairs. Producing training data of adequate quality proved complicated, and assembling a sufficient number of examples to cover realistic query patterns was impractical. Even after considerable effort, the model achieved poor precision, particularly for queries involving numerical conditions, and training a specialized model from scratch turned out to be expensive with limited benefit.

A second attempt moved away from fine-tuning and targeted SQT (which is described in Sec. 3.1) as an intermediate format for prompt-based generation. CQL (see Sec. 3.1) is a domain-specific language developed specifically for LinkAhead with no representation in general-purpose model training data, so we reasoned that SQT, being a structured JSON format, would be more familiar to language models and therefore easier to target reliably. Despite this, the models still struggled to reproduce the nested structure required for complex queries, particularly for numerical comparisons and multi-hop entity references. The underlying difficulty remained unchanged. Requiring the model to produce the output format in a single step places the full burden of structural and semantic correctness on the model, which it cannot reliably meet.

These findings motivated a different strategy based on decomposition and control. Rather than requiring the language model to produce the final query format in a single step, we introduced a simplified intermediate representation (QIR, see Sec. 3.2) designed to express the query in a clean, structured form that is easier for an LLM to generate reliably. The data schema plays an active role throughout. It is used to narrow the entities and properties provided to the LLM before generation, and to drive repair passes that correct common model errors in the output afterwards. The pipeline (see Sec. 3.3) begins with two focused LLM calls to identify the relevant entities and their properties from the schema, then generates the intermediate representation using that narrowed context. The output is then passed through a series of rule-based repair passes and validated against the schema. The resulting intermediate query is presented to the user for review and direct correction before it is deterministically converted to SQT and executed against the LinkAhead backend. The overall architecture is illustrated in Fig. 6.

3.1 CQL and Structured Query Trees (SQT)

CQL is the main query language used by LinkAhead. It is implemented in the server component and exposed via its API. This design allows its usage from GUIs as well as from batch processing scripts using LinkAhead's API clients. The domain-specific language follows a strict syntax and semantic which is parsed using an ANTLR parser [PQ95, Par13]. Its formal definition can be found in the server repository [Lin26b] and in the official documentation [Lin26c].

The example query

```
FIND Dataset WHICH IS REFERENCED BY Simulation
             WITH recording = 2015
```

can be used to find all `Records` of type `Dataset` which are linked from any property of another `Record` of type `Simulation`. Furthermore, the linking `Record` of type `Simulation` is required to have a property called `recording` with an exact value of 2015. In this example, keywords belonging to the formal CQL language are written in upper case, although this is not required by the specification. It can be seen that the language resembles English, but still requires a certain formal structure to be executed by the backend successfully.

In the server backend, CQL is translated to SQL which is used by the MariaDB database backend to process the query. One reason for choosing this design was to have a transparent processing in a prevalent language available, which can be e.g. used for debugging or profiling. The database backend itself makes use of a relational table structure that is described in [FSH+]. This structure maps LinkAhead's data model to a triple-store like structure which allows for flexible changes in schemas managed in LinkAhead.

For reducing ambiguities and for limiting the overhead caused by parsing CQL strings, a second query endpoint exists in the LinkAhead server that accepts SQTs in the form of JSON documents. These SQTs contain a parsed representation of a CQL string that can be directly converted to SQL.

The SQT for the example given in Sec. 3.1 is shown in the following listing. For easier readability we display the equivalent YAML representation instead of JSON[2].

```
selections: []
type: FIND
role: RECORD
versioned: false
entity:
  CQL_LEGACY:
    type: NORMAL
    pattern: Dataset
filter:
  REFERENCED_BY:
    asProperty: null
```

[2] This is a typesetting choice for print only. SQT is exchanged as JSON between LinkAhead components, and YAML is used here solely because it is more compact in a two-column layout. The equivalent JSON for this example is available in the released repository (`doc/examples/worked_example_sqt.json`, see Data availability).

```
    entity:
      CQL_LEGACY:
        type: NORMAL
        pattern: Simulation
    subProperty:
      SUBFILTER:
        filter:
          POV:
            property: recording
            operator: '='
            value: '2015'
            aggregate: ''
            subproperty: null
```

3.2 Query Intermediate Representation (QIR)

QIR is a compact JSON format designed as a simplified, LLM-friendly representation of the query structure. Unlike CQL, which requires mastery of a domain-specific syntax, and unlike SQT, whose structure closely mirrors the internal search engine protocol, QIR exposes only the fields necessary for query specification: a `verb` (`FIND`, `COUNT`, or `SELECT`), a `head_entity` naming the `RecordType` to query, and a `where` tree of filter conditions. The `where` tree is composed of a small set of typed nodes. `cond` nodes carry a property name, comparison operator, and value, `and`, `or`, and `not` nodes combine conditions using explicit argument lists, and `exists_ref` and `exists_backref` nodes express forward and backward entity references respectively. Every valid QIR maps deterministically to an SQT, so once the QIR passes schema validation, the resulting SQT is guaranteed to be structurally correct. For reference, we have added the full JSON schema for QIR to the appendix in Sec. A.

The QIR for the example given in Sec. 3.1 is given in the following listing. For easier readability we again display the equivalent YAML representation instead of JSON[3]. The example can also be seen in the graphical user interface in Fig. 2.

```
verb: FIND
head_entity: Dataset
where:
  type: exists_backref
  source_entity: Simulation
  source_prop: null
  where:
    type: cond
    prop: recording
```

[3] As with the SQT listing above, this is a print-readability choice only; the wire format LINDI produces and consumes is JSON, as reflected in the QIR JSON Schema in the Appendix (Sec. A). The equivalent JSON for the worked example in this section is available in the released repository (`doc/examples/worked_example_qir.json`, see Data availability).

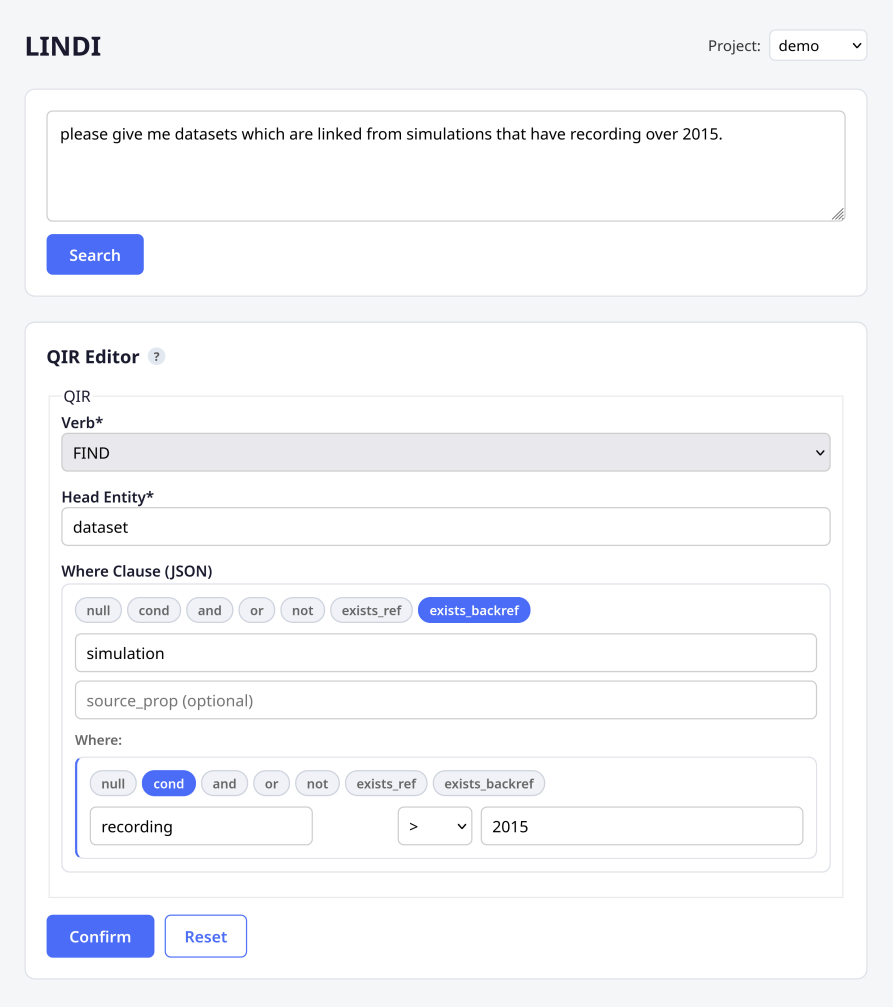


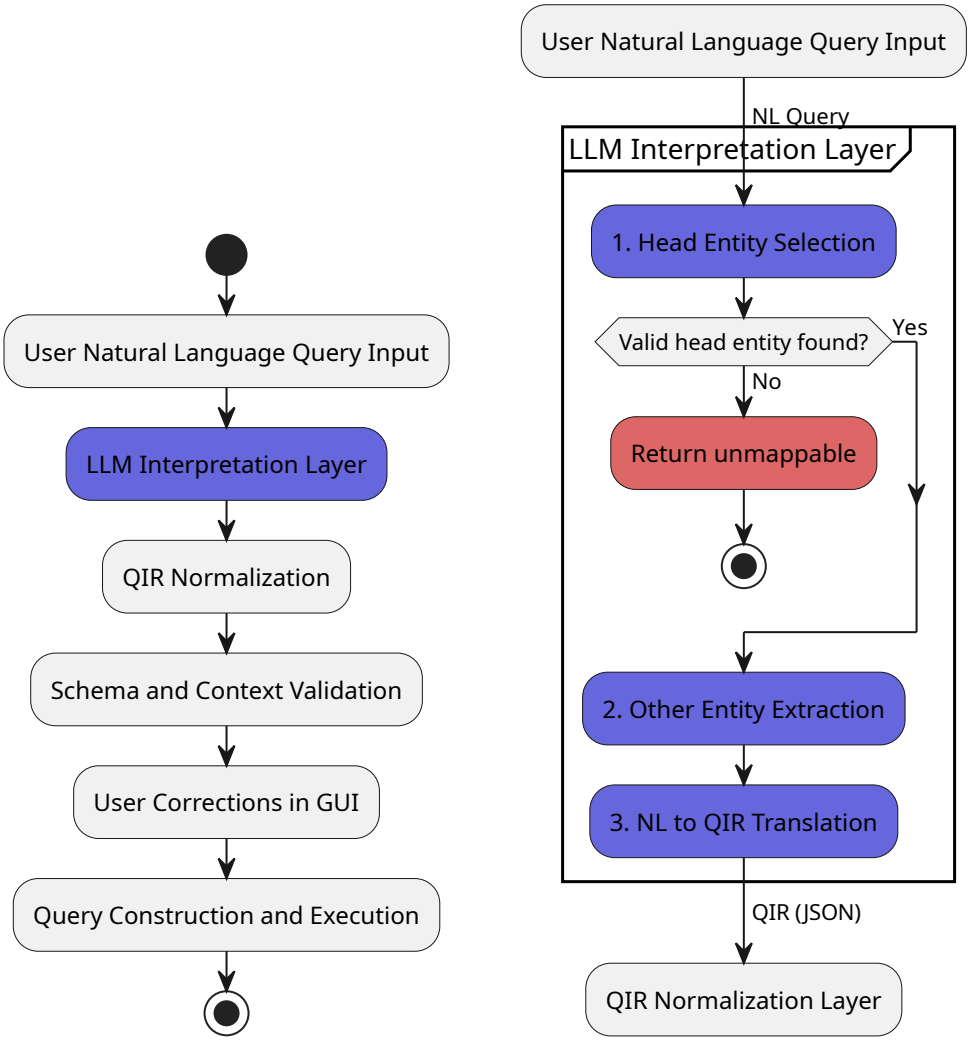
Figure 2: LINDI interface. The user enters a natural language query (top) and inspects or edits the generated QIR before confirming execution.

```
op: '='
val: '2015'
```

3.3 Translation Pipeline

The pipeline is structured in four layers. The overall control flow is shown in Figs. 3, 4 and 5.

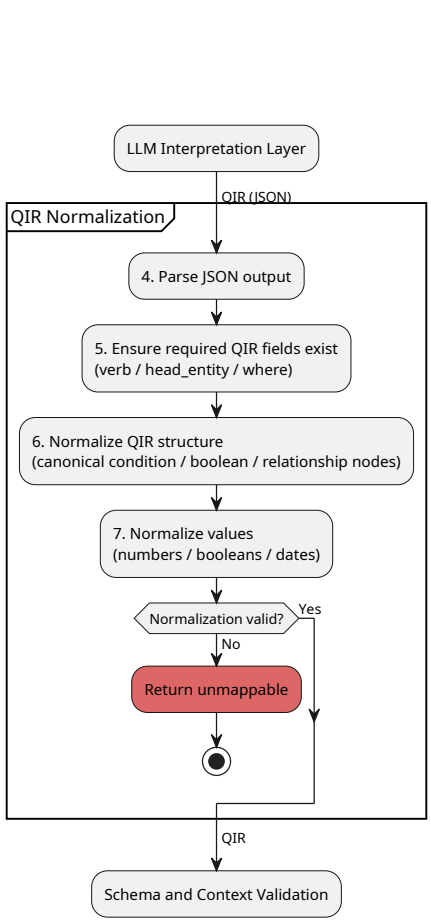
Layer 1: LLM interpretation (Fig. 3b). Two preparatory LLM calls narrow the schema before generation begins. The first identifies the head entity, the primary `RecordType` the query targets. If no valid head entity can be matched, the query is rejected immediately with a descriptive message. The second call extracts any additional entities referenced as filter constraints. The schema is then used to build a narrowed entity-property map covering only the selected entities,



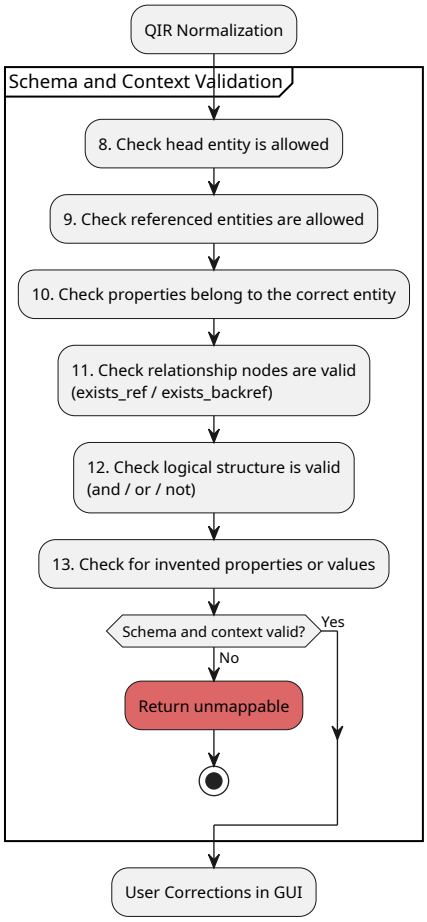
(a) Overview over the complete query pipeline from natural language input to structured query execution.

(b) LLM interpretation step. The model first identifies the head entity; queries with no valid head entity are rejected early, before any further processing is attempted. This step corresponds to layer 1 in Sec. 3.3.

Figure 3: Overview over the processing pipeline implemented in the LINDI-NL-to-SQT-Translator. The boxes in purple indicate steps where an LLM is used.



(a) QIR normalization step. Raw LLM output is parsed and brought into a canonical structure before reaching schema validation, catching format errors early and keeping the validator focused on semantic correctness. This step corresponds to layer 2 in Sec. 3.3.



(b) Schema validation step. The QIR is checked against the actual LinkAhead data schema, preventing hallucinated `RecordTypes` or `Properties` from reaching query execution. This step corresponds to layer 3 in Sec. 3.3.

Figure 4: Overview over the processing pipeline implemented in the LINDI-NL-to-SQT-Translator.

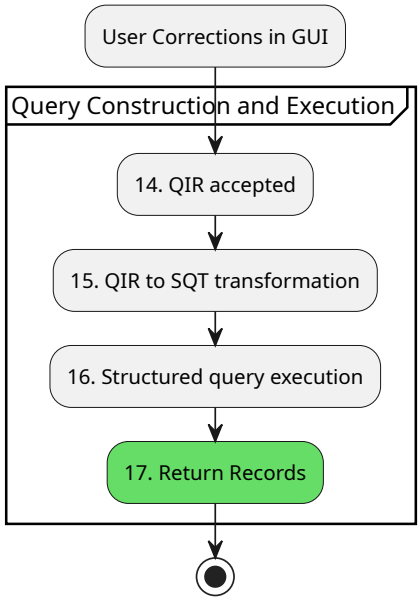


Figure 5: Query construction and execution step. A validated QIR is deterministically transformed into SQT and executed against the LinkAhead database. This corresponds to layer 4 in Sec. 3.3.

which is injected into the main generation prompt alongside few-shot examples retrieved by similarity to the input. The main LLM call produces a raw QIR JSON from this focused context.

Layer 2: QIR Normalization (Fig. 4a). The raw model output is passed through a sequence of deterministic repair passes that correct recurring model errors, including boolean shape mistakes, relationship direction errors, conditions attached to the wrong entity, and incomplete comparisons. The goal is to bring the output into a canonical, well-formed structure before it reaches the validator.

Layer 3: Schema and Context Validation (Fig. 4b). The normalized QIR undergoes two validation checks. First, a semantic check confirms that every property used in a condition is valid for the entity it is applied to at that nesting level, preventing hallucinated or misplaced properties from reaching query execution. Second, a structural check validates the QIR against the QIR JSON schema (see appendix Sec. A), ensuring the output is well-formed before conversion. Queries that fail either check return an unmappable response with a specific reason rather than silently producing a wrong query.

Layer 4: Query Construction and Execution (Fig. 5). A validated QIR is presented to the user via the LINDI interface in the LinkAhead WebUI (Fig. 2) for inspection and correction before execution. Because QIR is a compact, human-readable JSON format, users can review the translation and directly edit any field, for example to correct a misidentified entity type or

property name, without needing to understand CQL or SQT syntax. Once the user confirms the QIR, it is deterministically converted to SQT and submitted to the LinkAhead backend. (See Sec. 5 for why this human-in-the-loop step is part of the design.)

3.4 System Architecture

Fig. 6 shows how LINDI integrates with the existing LinkAhead infrastructure. The LinkAhead stack consists of three components:

- **Server** (Java): handles authentication, permissions, and SQT execution.
- **WebUI** (React): the web frontend through which users submit queries and inspect results. Furthermore, an editor for creating and updating the schema or `Records` is included.
- **Database** (MariaDB): stores all LinkAhead entities.

LINDI adds two components to this stack:

- **LINDI-NL-to-SQT-Translator** (FastAPI service): receives a natural language query together with the data schema, runs the translation pipeline, and returns either a QIR for user review or a converted SQT.
- **LINDI-Client** (Python library): runs as a subprocess inside the Server. It fetches and caches the schema, calls the translator, and routes responses back to the WebUI.

Furthermore, the WebUI was extended to include a component for submitting natural language queries and validating QIR. It is shown in Fig. 2.

3.5 Validation Datasets

We evaluate LINDI on four datasets, each paired with a real partner's LinkAhead schema: three in English (DS-1 myotwin, DS-2 zmt, DS-3 geomar) and one in German (DS-4 dimr). Each dataset is a YAML file pairing a natural-language query with its expected QIR, manually authored against that partner's actual schema (Sec. 4 reports per-dataset sample sizes and accuracy; see Data Availability for what is released).

DS-1 (myotwin, 71 queries) originates from myotwin GmbH, a biotech startup that develops a data-driven platform for cardiovascular research and drug development. The dataset is drawn from the joint IndiScale-myotwin project applying LinkAhead and LLM-assisted search to their production and research data [Ind25].

DS-2 (zmt, 44 queries) originates from the Leibniz Centre for Tropical Marine Research (ZMT) [HGO25], drawn from their institutional DataPortal for tracking marine-ecology research data.

DS-3 (geomar, 42 queries) originates from GEOMAR Helmholtz Centre for Ocean Research Kiel, drawn from their biosample management system for samples from expeditions and cruises [ME22].

DS-4 (dimr, 43 queries) originates from the Deutsches Institut für Menschenrechte (German Institute for Human Rights), drawn from their monitoring of measures against human trafficking in Germany [dim24]. Unlike the other three partners, no schema or query data from dimr is released; only this citation and the aggregate accuracy results are shared (see Data Availability).

4 Validation Results

We evaluated LINDI across four curated validation datasets with different data schemas, covering two natural languages (English and German). Each dataset was compiled manually and contains natural language queries paired with expected QIR outputs. All 200 queries were written by one of the authors, designed to approximate realistic user queries against each domain's actual schema rather than sampled from real usage logs. They were then checked and refined with LLM assistance rather than by a second human annotator. There is no overlap between the validation queries and the few-shot examples used during prompting in any of the four datasets. All accuracy numbers reported below are computed against these overlap-free datasets.

Accuracy is measured as the fraction of queries for which the system produces exactly the expected QIR. This is a deliberately conservative (strict) measure: any deviation from the expected QIR counts as a failure, even where the produced and expected QIR might be functionally equivalent. We reviewed the failure cases to distinguish plausible near-misses from genuine errors, but did not execute the mismatched QIRs against a live LinkAhead instance, so we cannot make that determination conclusively. Most failures are unambiguous, involving a different entity, property, or relationship than the one required to answer the query, but a few are plausible near-miss candidates worth flagging rather than silently counting as equally wrong; we give concrete examples alongside the failure-category breakdown below (Table 4). Loosening the strict measure is possible in at least two ways, each with a real cost: executing both QIRs against a live instance and comparing results directly, which risks false positives and negatives when the available data happens not to distinguish two logically different queries (e.g., no record sits exactly on a disputed date boundary); or a symbolic preprocessing step that checks logical equivalence on the QIR's structure itself, which can decide some cases (e.g., two representations of an equivalent value) without any data or execution at all, but not others where equivalence is a genuine data-dependent fact rather than a structural one. We leave both to future work and report the strict measure here.

Table 1 reports per-dataset and overall accuracy for two self-hosted open-weight models run without any fine-tuning or modification. Both models are served via Ollama on a single self-hosted machine (NVIDIA RTX 5090, 32GB VRAM; 62GB RAM), at temperature 0.0 for deterministic output. We ran the full evaluation 3 times per dataset and model to check this determinism empirically. Llama 3.1 (8B) was perfectly stable across all 12 runs (zero spread on every dataset); GPT-OSS (20B) was likewise stable on three of the four datasets, with only DS-1 (myotwin) showing a small spread (91.5%/91.5%/90.1%, 1.4 percentage points). Overall accuracy is computed as the weighted mean across all datasets (total correct queries divided by total queries).

On the three English-language datasets (DS-1 to DS-3), Llama 3.1 (8B) [D+24] achieves accuracy in the range 90.1% to 97.7%, while GPT-OSS (20B) [Ope25] is comparatively tighter

Table 1: Validation accuracy by dataset and model. DS-1 to DS-3 contain English queries; DS-4 contains German queries. Overall accuracy is the weighted mean across all 200 queries.

Dataset	Lang.	Queries	Llama 3.1 (8B)	GPT-OSS (20B)
DS-1 (myotwin)	EN	71	90.1%	91.5%
DS-2 (zmt)	EN	44	97.7%	93.2%
DS-3 (geomar)	EN	42	97.6%	92.9%
DS-4 (dimr)	DE	43	65.1%	58.1%
Overall		**200**	**88.0%**	**85.0%**

(91.5% to 93.2%). The German-language dataset (DS-4) reveals a pronounced language dependency for both models. Llama 3.1 (8B) drops to 65.1%, while GPT-OSS (20B) drops even further to 58.1%, despite GPT-OSS's broader multilingual training. Overall accuracy is 88.0% for Llama 3.1 (8B) and 85.0% for GPT-OSS (20B). Llama 3.1 (8B) leads on three of the four datasets (DS-2, DS-3, DS-4), with GPT-OSS (20B) ahead only on DS-1 (myotwin).

Accuracy stays consistently high across all three English-language datasets, regardless of scientific domain. This suggests that schema-aware context narrowing transfers well to new schemas, and that performance depends more on the query language than on differences between schemas. GPT-OSS (20B)'s larger drop on German also coincides with a reproducible slow-generation pattern, where a fixed subset of German queries consistently exceeds the generation-time budget on every repeat (Table 4). German-language performance for both models, and why it underperforms relative to English, is discussed in Sec. 5.2.

The results above show that the full pipeline achieves high accuracy, but they do not by themselves show whether decomposing translation into narrower, schema-aware steps is what makes this possible, as opposed to simply prompting a modern LLM directly. We test this claim with two experiments: a single-step baseline and a per-stage ablation, both run on the same 200 queries and exact-match harness as Table 1. We ran each dataset/model/variant combination once; we did not repeat these runs, since the stability check above already showed results vary little across repeats.

Single-step baseline. Table 2 replaces the full pipeline's three LLM calls, schema narrowing, and repair passes with a single LLM call given the full (un-narrowed) schema and asked to produce the AST directly. The drop is large and consistent for Llama 3.1 (8B) on every dataset (-2.4 to -44.2 points, largest on German). For GPT-OSS (20B), the baseline stays close to the full pipeline on the three English datasets: a -2.8-point drop on myotwin, no change on zmt, and a -4.8-point drop on geomar. But on German (DS-4) it collapses even more severely than Llama's (58.1% to 9.3%, a -48.8 point drop). The single-step prompt essentially never recovers a valid German compound entity name without narrowing's help.

Per-stage ablation. Table 3 disables one pipeline stage at a time from the full pipeline: schema-aware entity narrowing, the deterministic repair passes, or the final validation gate. Disabling validation never changes accuracy for Llama 3.1 (8B) (exact match to the full pipeline on every dataset); this is expected, since validation can only convert a wrong answer into an explicit rejection, never turn a wrong answer into a correct one. Repairs are the dominant factor for Llama 3.1 (8B) on the English datasets (up to -15.9 points), but narrowing dominates on German

Table 2: Single-step baseline vs. the full pipeline, one run per dataset/model.

Dataset	Queries	Llama full	Llama 1-step	GPT-OSS full	GPT-OSS 1-step
DS-1 (myotwin)	71	90.1%	80.3%	91.5%	88.7%
DS-2 (zmt)	44	97.7%	79.5%	93.2%	93.2%
DS-3 (geomar)	42	97.6%	95.2%	92.9%	88.1%
DS-4 (dimr)	43	65.1%	20.9%	58.1%	9.3%
Overall	**200**	**88.0%**	**70.5%**	**85.0%**	**72.5%**

(-27.9 points from narrowing vs. only -2.3 from repairs), the opposite pattern, and a genuine cross-language difference. For GPT-OSS (20B), disabling narrowing has little effect on the English datasets: no change on zmt (93.2% either way) and a small -2.4-point drop on geomar, both far smaller than the -32.6-point drop on German. On German (DS-4), disabling narrowing brings accuracy down to 25.6%, still well above the single-step baseline (9.3%), confirming that repairs and validation retain real value even with narrowing disabled, while narrowing remains the larger of the two contributors on this dataset (-32.6 points vs. -25.5 for disabling repairs).

Table 3: Per-stage ablation: accuracy with one pipeline stage disabled at a time (delta vs. the full pipeline in parentheses).

	Llama 3.1 (8B)		
Dataset	**no narrowing**	**no repairs**	**no validation**
DS-1 (myotwin)	88.7% (-1.4)	77.5% (-12.6)	90.1% (0.0)
DS-2 (zmt)	95.5% (-2.2)	81.8% (-15.9)	97.7% (0.0)
DS-3 (geomar)	97.6% (0.0)	92.9% (-4.7)	97.6% (0.0)
DS-4 (dimr)	37.2% (-27.9)	62.8% (-2.3)	65.1% (0.0)
Overall	**81.0% (-7.0)**	**78.5% (-9.5)**	**88.0% (0.0)**
	GPT-OSS (20B)		
Dataset	**no narrowing**	**no repairs**	**no validation**
DS-1 (myotwin)	90.1% (-1.4)	85.9% (-5.6)	90.1% (-1.4)
DS-2 (zmt)	93.2% (0.0)	93.2% (0.0)	93.2% (0.0)
DS-3 (geomar)	90.5% (-2.4)	88.1% (-4.8)	92.9% (0.0)
DS-4 (dimr)	25.6% (-32.6)	32.6% (-25.5)	58.1% (0.0)
Overall	**77.0% (-8.0)**	**76.5% (-8.5)**	**84.5% (-0.5)**

The GPT-OSS/myotwin -1.4-point figures under no-validation and no-narrowing are not a real effect of those ablations; they reflect the same small run-to-run variance already noted in the stability check (91.5%/91.5%/90.1% across repeats), landing on the lower run this time. On DS-4 (dimr), disabling validation makes no difference for GPT-OSS (20B) either (0.0), consistent with the pattern already established for Llama 3.1 (8B) on all four datasets and for GPT-OSS (20B) on the other three, since validation is a strict post-hoc filter, it cannot logically improve on an already-generated answer. This holds specifically when the expected answer is not itself `unmappable`; a wrong answer that validation catches and rejects would only match a legitimately-unmappable expected answer, a case that does not arise for this model/dataset com-

bination. Disabling repairs, by contrast, does lower accuracy substantially (-25.5 points), though less severely than disabling narrowing (-32.6 points), reinforcing that narrowing is the dominant factor for this model on this dataset.

Taken together, decomposition clearly helps on average and substantially so for Llama on English and for both models on German narrowing specifically, supporting the paper's central claim; disabling narrowing never improves accuracy for either model on any dataset, though the size of its contribution varies widely by language and dataset.

The observed failure modes across both models fall into several recurring categories:

- Property attachment errors when conditions follow a relationship clause
- Incorrect comparison operators for ambiguous phrases (e. g. *above* interpreted as $\geq$ instead of $>$)
- Entity selection errors for compound or ambiguous entity names
- Errors when property values contain multiple words
- Missed rejection of queries outside the system's supported scope
- Hallucinated conditions not present in the original query

Table 4 reports counts and percentages per category and per model, based on one representative run per dataset (the stability check above shows the pipeline is deterministic on 11 of the 12 model/dataset pairs). Entity selection errors dominate for both models and are concentrated in the German dataset (DS-4). The incorrect-comparison-operator failures, which affect only Llama 3.1 (8B) (4 cases), are a single recurring near-miss pattern rather than four unrelated errors, all traceable to an inclusive-vs.-exclusive date-boundary ambiguity on "after"/"nach" phrasing that could execute identically to the expected QIR depending on how LinkAhead's query backend handles date-prefix matching. Two further individual near-miss candidates were also observed, not broken out as separate rows in the table. One case involved a numeric value returned as the string `"2022"` instead of the integer `2022`, and another gave a month as `"2024-07"` instead of an explicit start/end date range; both could likewise execute identically to the expected QIR depending on LinkAhead's type coercion. Multi-word property values affect only GPT-OSS (20B). A residual category not covered by the six above (labelled "Other") accounts for JSON-formatting/parsing errors and, for GPT-OSS (20B) specifically, the cluster of 7 German queries discussed above that reproducibly exceeded the generation-time budget.

All of these trace back to the same underlying issue, namely the LLM's incomplete resolution of the semantic structure of the natural language input. Users can reduce mistranslations by phrasing queries with explicit entity names, unambiguous comparison terms, and clearly bounded string values.

5 Discussion

The modular pipeline design of LINDI addresses the core reliability challenge identified during development. No single LLM call produces the final query format reliably, but a sequence of

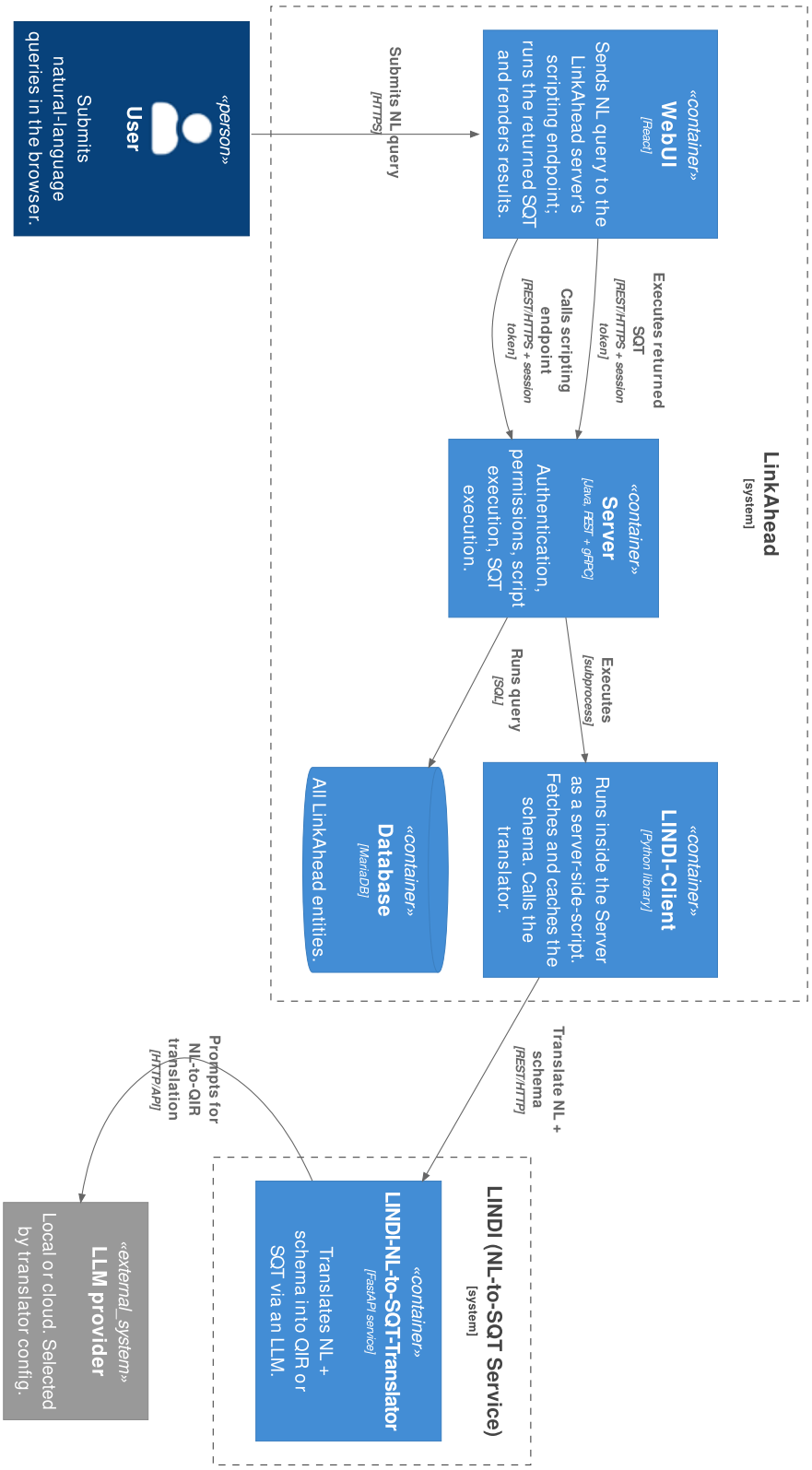


Figure 6: Architecture of the integration of LINDI into the LinkAhead system context. LINDI runs as an internal service, keeping the LLM interface isolated from direct user access.

Table 4: Failure counts by category, from one representative run per dataset (24 failures for Llama 3.1 8B, 30 for GPT-OSS 20B, out of 200 queries each).

Category	**Llama 3.1 (8B)**		**GPT-OSS (20B)**	
	count	%	count	%
Property attachment after relationship	3	12.5%	3	10.0%
Incorrect comparison operators	4	16.7%	0	0.0%
Entity selection (compound/ambiguous names)	10	41.7%	10	33.3%
Multi-word property values	0	0.0%	1	3.3%
Missed rejection of out-of-scope queries	1	4.2%	1	3.3%
Hallucinated conditions	2	8.3%	5	16.7%
Other (formatting errors, generation timeouts)	4	16.7%	10	33.3%

focused calls [WWS+22], each operating over a narrowed context, yields consistent results. Introducing QIR as an intermediate representation serves a dual purpose. It gives the LLM a compact, constrained target that is easier to generate correctly than CQL or SQT, and it gives the user a human-readable artifact that can be corrected directly without knowledge of the underlying query languages. Schema-aware context narrowing limits the surface area exposed to the model, while rule-based normalization and validation layers deterministically correct systematic model errors and enforce schema constraints after generation. Delegating these corrections to explicit rules rather than to the model itself improves translation accuracy without retraining, and makes the failure modes of the system predictable and auditable. The result is a system that keeps the LLM's role narrow and makes its errors recoverable rather than silent.

The pipeline's modular structure also has practical deployment benefits. Each layer has a well-defined interface, so individual components can be improved or replaced independently. For instance, the LLM can be swapped for a newer model, repair passes can be extended to handle newly observed error patterns, and schema validation evolves without touching the generation step. Because the prompts are narrow and focused, the system is compatible with small, self-hosted open-source or open-weight models [DG24], avoiding dependence on large proprietary LLMs and the associated data-privacy and deployment constraints.

While CQL requires the user to write a full query from scratch in an unbounded grammar, QIR is instead generated by the pipeline and exposes only a `verb`, a `head_entity`, and a small `where` tree of six node types (Sec. 3.2), so the user's task becomes checking a short draft against their intent rather than writing correct syntax. Because QIR is schema-validated before execution, a QIR that passes validation is guaranteed to produce a structurally correct SQT, and it remains directly inspectable and editable in the LINDI GUI (Fig. 2) rather than an opaque intermediate. This maps onto the recognition-over-recall principle [Nie94]. CQL demands recalling syntax from an unbounded grammar, while QIR only requires recognizing and verifying a bounded set of node types. This is a stronger basis than an unsupported design assertion, but it remains a design rationale rather than a proven usability result, since no controlled user study comparing QIR and CQL has been conducted (Sec. 5.2).

Although LINDI targets LinkAhead specifically, its broader relevance rests on two distinct claims. The narrower claim is that the implementation could port to another platform, since

QIR's node types express a generic typed-entity-with-references model, not a LinkAhead-specific one, and generalizing would require rebuilding three parts: the deterministic QIR-to-target-query conversion step, currently written for LinkAhead's SQT format; the schema introspection that builds the narrowed entity-property map, currently reading LinkAhead's own schema API; and the prompts and few-shot examples, currently tuned to LinkAhead-specific vocabulary. The schema-aware narrowing, repair-pass, and human-in-the-loop review architecture is not LinkAhead-specific and should carry over largely unchanged, though this is architectural reasoning, not a claim tested by actually porting the pipeline to another system. The broader claim, that the underlying pattern itself is not LinkAhead-specific to begin with, is addressed in this section's closing paragraph.

5.1 Model Capabilities

LINDI supports three query verbs: **FIND** (record retrieval), **COUNT** (counting matching records), and **SELECT** (projecting specific property values). Within these verbs, the system handles:

- Entity recognition, including plural forms and casual or informal phrasing
- Property filtering with equality and numeric comparison operators
- Logical conditions using AND and OR
- Forward and backward relationship queries (`exists_ref` and `exists_backref`)
- Date normalization to ISO format
- Nested conditions combining relationship filters and property filters

5.2 Model Limitations

The following query types fall outside the system's supported scope and are explicitly rejected with a descriptive error message:

- Data modification operations (`DELETE`, `UPDATE`, `INSERT`)
- References to entities or properties not defined in the schema
- Mathematical expressions in values of conditions (e. g. $cell_count/volume > 1000$)
- Superlative queries (e. g. *highest*, *most recent*)
- Queries with ambiguous head entity selection
- Numeric values that require unit conversion

Two further boundaries of QIR's expressiveness are not yet covered by this list and should be stated explicitly. First, relationship traversal is limited to two hops; a third-level relationship chain is not supported. Second, the schema validator cannot always distinguish a legitimate

"property exists" check (`op: null, val: null`) from a malformed or hallucinated condition of the same shape, most likely to arise from query types the system is not designed to handle (e.g., the mathematical-expression case above); such cases are not currently guaranteed to be caught and rejected. As for how failures are surfaced to the user, queries that fail schema or structural validation return an `unmappable` response with a specific reason rather than executing (Sec. 3.3, Layer 3); queries that pass validation are shown to the user as an editable QIR in the LINDI interface before execution (Sec. 3.3, Layer 4).

As already noted above, this remains an open limitation rather than a measured result. Our evaluation is designed to simulate a real use-case: hand-crafted queries against real partner data models, with validation accuracy serving as a proxy for how often users need to correct a draft QIR before running it. Every query where the generated QIR does not exactly match the expected one is, by construction, a case where a user would have had to edit or reject the draft. On the three English-language datasets, that puts the correction rate around 2.3%–9.9% depending on model and dataset (mirroring 90.1%–97.7% accuracy); on the German dataset it is substantially higher, 34.9%–41.9% depending on model. This is an upper bound, not a direct usage measurement, and does not distinguish a full rewrite from a one-field edit. A genuine accessibility study, analyzing real usage logs, evaluating user experience, or assessing the UI itself, is beyond the scope of this article; we report this proxy as the closest approximation available within it.

As Sec. 4 discusses, German-language accuracy for both models remains far below English, and neither model choice is a reliable lever for closing that gap. English was also the pipeline's primary development target throughout, with prompts and few-shot examples iterated against English data, and the German dataset was added afterward to test how far that English-tuned pipeline generalizes, not as a co-equal design goal from the start. The ablations above localize why it falls short, since narrowing matches query wording against schema vocabulary that is itself in English, so a German query must clear a matching step that was never tuned for German, while the remaining pipeline stages (decomposed LLM calls, repairs, validation) are language-agnostic by construction. Closing this gap with dedicated German-language tuning is left as future work. The accessibility case for non-technical users rests on the QIR design rationale given earlier in this section (recognition over recall), not on accuracy alone. An editable, schema-validated draft is equally checkable regardless of language, so German-language accuracy is a separate axis from whether the interaction model itself is accessible.

Per-dataset sample sizes are also modest: 42 (geomar) to 71 (myotwin) English queries, and 43 for the German dataset (dimr). Each dataset's queries were manually authored against a real partner schema rather than sampled from usage logs. Larger, partner-sourced or usage-log-based datasets are left as future work. Results at the per-dataset level should be read as descriptive rather than statistically definitive.

Several challenges remain open. The LLM landscape is evolving rapidly, and prompt behavior is sensitive to model version, which makes long-term prompt stability a practical concern for any deployed system of this kind. Natural language ambiguity is the most fundamental constraint, and is directly reflected in the observed failure modes. Ambiguous comparison phrases, compound or informal entity names, multi-word string values, and underspecified queries all produce mistranslations that the system cannot fully resolve without user correction. Hallucinations [JLF+23], where the model generates plausible-sounding but incorrect properties or conditions, arise from the same root cause. The model fills gaps in underspecified input with confident but

wrong inferences.

Positioned against the prior work discussed in Sec. 2, LINDI's contribution is deliberately applied rather than methodological. It does not introduce a new translation method, but demonstrates two concrete instantiation choices. The QIR node design, in particular `exists_ref` and `exists_backref`, is built specifically for LinkAhead's bidirectional reference model, which is neither relational-join-shaped nor generic RDF. And the pipeline is validated end-to-end across four independent real-world schemas and two languages, rather than on a single benchmark domain.

Acknowledgements: IndiScale GmbH acknowledges funding from NBank in the program "Innovationsförderprogramm für FuE in Unternehmen" which is co-financed by the European Union. The name of the project is "AI-based optimization of the production of cardiac bodies" [Ind25]. We thank our collaboration partners Hendrik Windel, Maria Gaysaeva and Jan Pietras from myotwin GmbH and the whole team at IndiScale GmbH for fruitful and constructive discussions; Furthermore, we thank myotwin GmbH, the Leibniz Centre for Tropical Marine Research (ZMT), GEOMAR Helmholtz Centre for Ocean Research Kiel, and the Deutsches Institut für Menschenrechte for providing data schemas.

Data availability: LinkAhead is open-source software and available under AGPLv3 on gitlab.com [Lin26a].

LINDI consists of three software components which are released as open-source software, here:

- LINDI-Translator: https://gitlab.indiscale.com/caosdb/src/lindi-translator
- LINDI-Client: https://gitlab.indiscale.com/caosdb/src/lindi-client
- LINDI-GUI: https://gitlab.indiscale.com/caosdb/src/lindi-gui

The LINDI-Translator also includes prompts, JSON schemas, and the validation harness.

All datasets and validation queries used in Table 1 are published in the released GitLab repository for the validation code: https://gitlab.indiscale.com/caosdb/src/lindi-validation.

An exception is DS-4 (dimr), for which we do not have permission to release any data; aggregate accuracy results for DS-4 are reported in Table 1.

Declaration on the use of AI: During the creation of this article, AI was used for the following tasks:

- Assistance for reviewing LaTeX code.
- Language checks (grammar, style, typography).
- Assistance with `plantuml` for creating the diagrams in Figs. 6,3,4,5.
- Assistance with batch processing for creating Tab. 1. This procedure was checked manually.

- Assistance with checking and refining the validation dataset queries and their expected QIR outputs (Sec. 4), which were authored by one of the authors.

All suggestions generated by the AI were reviewed and edited by the authors before they were integrated into the article. Mostly claude.ai with the Opus 4.7 model was used for these tasks. The software described in the article is based on LLMs which implies that AI was involved in obtaining the results. Some parts of the software described in the article were generated with AI, mostly with claude.ai with the Opus model series. This code is subject to quality control including human code review.

Bibliography

[Ant24] Anthropic. Introducing the Model Context Protocol. Blog post, Nov. 2024. Last accessed on 2026-05-20.
https://www.anthropic.com/news/model-context-protocol

[BCP+26] Y. Bengio, S. Clare, C. Prunkl, M. Andriushchenko, B. Bucknall, M. Murray, R. Bommasani, S. Casper, T. Davidson, R. Douglas, D. Duvenaud, P. Fox, U. Gohar, R. Hadshar, A. Ho, T. Hu, C. Jones, S. Kapoor, A. Kasirzadeh, S. Manning, N. Maslej, V. Mavroudis, C. McGlynn, R. Moulange, J. Newman, K. Y. Ng, P. Paskov, S. Rismani, G. Sastry, E. Seger, S. Singer, C. Stix, L. Velasco, N. Wheeler, D. Acemoglu, V. Conitzer, T. G. Dietterich, F. Heintz, G. Hinton, N. Jennings, S. Leavy, T. Ludermir, V. Marda, H. Margetts, J. McDermid, J. Munga, A. Narayanan, A. Nelson, C. Neppel, S. D. Ramchurn, S. Russell, M. Schaake, B. Schölkopf, A. Soto, L. Tiedrich, G. Varoquaux, A. Yao, Y.-Q. Zhang, L. A. Aguirre, O. Ajala, F. Albalawi, N. AlMalek, C. Busch, J. Collas, A. C. P. de Leon Ferreira de Carvalho, A. Gill, A. H. Hatip, J. Heikkilä, C. Johnson, G. Jolly, Z. Katzir, M. N. Kerema, H. Kitano, A. Krüger, K. M. Lee, J. R. L. Portillo, A. McLysaght, O. Molchanovskyi, A. Monti, M. Nemer, N. Oliver, R. Pezoa, A. Plonk, B. Ravindran, H. Riza, C. Rugege, H. Sheikh, D. Wong, Y. Zeng, L. Zhu, D. Privitera, S. Mindermann. International AI Safety Report 2026. 2026.
https://arxiv.org/abs/2602.21012

[D+24] A. Dubey et al. The Llama 3 Herd of Models. 2024.
https://arxiv.org/abs/2407.21783

[DCZ22] N. Deng, Y. Chen, Y. Zhang. Recent Advances in Text-to-SQL: A Survey of What We Have and What We Expect. In *Proceedings of the 29th International Conference on Computational Linguistics*. Pp. 2166–2187. International Committee on Computational Linguistics, Gyeongju, Republic of Korea, oct 2022.
https://aclanthology.org/2022.coling-1.190/

[DG24] A. Dasgupta, D. Guha. A Literature Survey on Open Source Large Language Models. In *Proceedings of the 2024 7th International Conference on Computers in Management and Business*. Association for Computing Machinery, Singapore, jan 2024.

doi:10.1145/3647782.3647803
https://dl.acm.org/doi/abs/10.1145/3647782.3647803

[dim24] *Monitor Menschenhandel in Deutschland: Erster Periodischer Bericht*. Bericht / Deutsches Institut für Menschenrechte. Deutsches Institut für Menschenrechte, Berichterstattungsstelle Menschenhandel, Berlin, 2024.
https://nbn-resolving.org/urn:nbn:de:0168-ssoar-101336-4

[Fre24] J. Freitag. Ice sample and data management with CaosDB in the Glaciology section at AWI. In *Workshop on Applications of Semantic Data Management (ASDM 2024)*. Göttingen, Germany, apr 2024.

[FSH+] T. Fitschen, A. Schlemmer, D. Hornung, H. tom Wörden, U. Parlitz, S. Luther. CaosDB—Research Data Management for Complex, Changing, and Automated Research Workflows. 4(2):83.
doi:10.3390/data4020083
https://www.mdpi.com/2306-5729/4/2/83

[HGO25] B. Hemmelskamp-Pfeiffer, H. C. George, F. Opätz. Towards Institutional Data Tracking: A Case Study from the Leibniz Centre for Tropical Marine Research (ZMT). 9 2025.
doi:10.5281/zenodo.17223681

[HSW] D. Hornung, F. Spreckelsen, T. Weiß. Agile Research Data Management with Open Source: LinkAhead. 1(2). Medium: application/pdf,application/xml Publisher: Universitäts- und Landesbibliothek Darmstadt.
doi:10.48694/INGGRID.3866
https://www.inggrid.org/article/id/3866/

[Ind25] IndiScale. Project Website: AI-based optimization of the production of cardiac bodies. Website, May 2025. Last accessed on 2026-05-20.
https://www.indiscale.com/projects/ai-based-optimization-of-the-production-of-cardiac-bodies/

[JLF+23] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, P. Fung. Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys* 55(12):248, 2023.
doi:10.1145/3571730
https://dl.acm.org/doi/10.1145/3571730

[Lin26a] LinkAhead Developers. LinkAhead Gitlab Group. Gitlab, May 2026. Last accessed on 2026-05-18.
https://gitlab.com/linkahead/

[Lin26b] LinkAhead Developers. LinkAhead Server Repository. Gitlab, May 2026. Last accessed on 2026-05-18.
https://gitlab.com/linkahead/linkahead-server/

[Lin26c] LinkAhead Developers. Official LinkAhead Documentation. Website, May 2026. Last accessed on 2026-05-18.
http://docs.linkahead.org

[LS26] S. T. Lennartz, A. Schlemmer. FAIR Workflows in Earth System Modelling: A Use Case With Semantic Data Management. *Geoscience Data Journal* 13(1):e70055, 2026.
doi:10.1002/gdj3.70055

[ME22] F. Mittermayer, J. Eckstein. SOP Biosamplemanagement for samples from expeditions and cruises at GEOMAR. 8 2022.
doi:10.5281/zenodo.7003999

[MRO+25] A. Meloni, D. Reforgiato Recupero, F. Osborne, A. Salatino, E. Motta, S. Vahdati, J. Lehmann. Exploring Large Language Models for Scientific Question Answering via Natural Language to SPARQL Translation. *ACM Transactions on Intelligent Systems and Technology*, 2025.
doi:10.1145/3757923

[Nie94] J. Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*. Pp. 152–158. 1994.
doi:10.1145/191666.191729

[Ope25] OpenAI. gpt-oss-120b & gpt-oss-20b Model Card. 2025.
https://arxiv.org/abs/2508.10925

[Par13] T. Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2013.

[PC26] G. Papyshev, K. J. D. Chan. AI regulatory strategies for digital sovereignty: The role of geopolitics and technological disparities. *Electronic Markets* 36(1):8, 2026.
doi:10.1007/s12525-025-00870-z

[PG24] R. Patil, V. R. Gudivada. A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs). *Applied Sciences* 14(5):2074, 2024.
doi:10.3390/app14052074
https://www.mdpi.com/2076-3417/14/5/2074

[PQ95] T. J. Parr, R. W. Quong. ANTLR: A predicated-LL (k) parser generator. *Software: Practice and Experience* 25(7):789–810, 1995.
doi:10.1002/spe.4380250705

[RWC+19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever. Language Models are Unsupervised Multitask Learners. Technical report, OpenAI, 2019.
https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[SSTW25] A. Schlemmer, F. Spreckelsen, F. Tristram, H. tom Wörden. Bridging Repositories, ELNs and Semantic Data Management: A LinkAhead-based use case for 3D Additive Manufacturing. Apr. 2025.
doi:10.5281/zenodo.15174573
https://doi.org/10.5281/zenodo.15174573

[SWF+19] A. Schlemmer, H. tom Wörden, J. Freitag, T. Fitschen, J. Kerch, Y. Schlomann, D. Hornung, I. Weikusat, U. Parlitz, F. Wilhelms et al. Evaluation of the semantic research data management system CaosDB in glaciology. In *Conference for Research Software Engineers in Germany (deRSE19)*. Potsdam, Germany, jun 2019.
doi:10.5446/42478

[WSL+] H. tom Wörden, F. Spreckelsen, S. Luther, U. Parlitz, A. Schlemmer. Mapping Hierarchical File Structures to Semantic Data Models for Efficient Data Integration into Research Data Management Systems. 9(2):24.
doi:10.3390/data9020024
https://www.mdpi.com/2306-5729/9/2/24

[WWS+22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*. Volume 35, pp. 24824–24837. Curran Associates, Inc., 2022.
https://arxiv.org/abs/2201.11903

[YHY+24] J. Yang, B. Hui, M. Yang, J. Yang, J. Lin, C. Zhou. Synthesizing Text-to-SQL Data from Weak and Strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. 2024.
doi:10.18653/v1/2024.acl-long.425

[YZY+18] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, D. Radev. 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*. Pp. 3911–3921. Association for Computational Linguistics, Brussels, Belgium, oct 2018.
doi:10.18653/v1/D18-1425
https://aclanthology.org/D18-1425/

A JSON Schema for QIR

The following listing shows a YAML representation of the JSON schema for QIR.

```
$defs:
  cond:
    type: object
    properties:
      type: {type: string, const: cond}
      prop: {type: string}
      op:
        type: [string, "null"]
        enum: ["=", "<", ">", "<=", ">=", "!=", "like", null]
      val: {type: [string, number, boolean, "null"]}
  and:
    type: object
    properties:
      type: {type: string, const: and}
      args:
        type: array
        items:
          anyOf: &id_items
            - $ref: "#/$defs/cond"
            - $ref: "#/$defs/and"
            - $ref: "#/$defs/or"
            - $ref: "#/$defs/not"
            - $ref: "#/$defs/exists_ref"
            - $ref: "#/$defs/exists_backref"
  or:
    type: object
    properties:
      type: {type: string, const: or}
      args:
        type: array
        items:
          anyOf: *id_items
  not:
    type: object
    properties:
      type: {type: string, const: not}
      arg:
        type: array
        minItems: 1
        maxItems: 1
        items:
          anyOf: *id_items
  exists_ref:
    type: object
    properties:
      type: {type: string, const: exists_ref}
```

```
      target_entity: {type: string}
      via_prop: {type: [string, "null"]}
      where:
        anyOf: &id_items_null
          - $ref: "#/$defs/cond"
          - $ref: "#/$defs/and"
          - $ref: "#/$defs/or"
          - $ref: "#/$defs/not"
          - $ref: "#/$defs/exists_ref"
          - $ref: "#/$defs/exists_backref"
          - type: "null"
  exists_backref:
    type: object
    properties:
      type: {type: string, const: exists_backref}
      source_entity: {type: string}
      source_prop:
        type: [string, "null"]
      where:
        anyOf: *id_items_null
  select_prop:
    type: object
    properties:
      prop: {type: string}
      sub_prop:
        type: [string, "null"]
    required: [prop]

$schema: https://json-schema.org/draft/2019-09/schema
type: object
properties:
  verb:
    enum: [FIND,SELECT,COUNT]
    type: string
  head_entity: {type: string}
  select_props:
    type: array
    items:
      $ref: "#/$defs/select_prop"
  where:
    anyOf: *id_items_null
additionalProperties: false
```