Ontology-Grounded Large Language Models for Reliable Querying of Wind Turbine Inspection Knowledge

Louis Verstraeten[1], Xavier Chesterman[1], Jan Helsen[1], and Ann Nowé[1]

[1] *Vrije Universiteit Brussel, Brussels, Belgium*
Louis.Verstraeten@vub.be
Xavier.Chesterman@vub.be
Jan.Helsen@vub.be
Ann.Nowe@ai.vub.ac.be

ABSTRACT

Inspection reports of industrial assets contain valuable diagnostic knowledge, but their unstructured nature makes automated reasoning difficult. This paper presents an ontology-grounded question answering framework for querying wind turbine gearbox inspection reports using natural language. Inspection data are automatically parsed into structured representations consisting of a domain ontology and a knowledge graph. On top of this representation, a large language model translates user questions into SPARQL queries. To improve robustness, we employ example-based query generation combined with an Ontology-Based Query Checker (OBQC) that validates generated queries against ontology constraints and iteratively repairs violations before execution. The approach is evaluated on real-world inspection reports using 50 diagnostic prompts of varying complexity, achieving a 96% successful execution rate. Results demonstrate that combining ontology grounding with constrained LLM-based query generation enables reliable and flexible diagnostic reasoning over inspection documentation.

1. INTRODUCTION

Predictive maintenance has become a central strategy in Industry 4.0 because it reduces unplanned downtime, improves asset utilization, and lowers repair costs (Achouch et al., 2022). In wind energy systems, predictive maintenance is particularly important because turbines are often deployed in remote or offshore locations where maintenance requires specialized vessels and is constrained by weather windows. In this paper, we focus on gearbox inspection as a representative and high-impact maintenance use case to ground the proposed approach. In addition, operation and maintenance activities account for a significant share of wind farm lifecycle costs, meaning that unexpected component failures can lead to substantial financial losses. Finally, turbines operate under highly variable mechanical loads and harsh environmental conditions, which accelerate component degradation and make early fault detection essential for maintaining availability and energy production.

Although condition monitoring pipelines have advanced substantially, a large share of actionable diagnostic evidence still appears in unstructured technical text, including borescope observations, service notes, and maintenance actions. Prior work on technical language supervision shows that these textual artifacts encode rich fault semantics and can support intelligent fault diagnosis when properly modeled (Löwenmark, Taal, Schnabel, Liwicki, & Sandin, 2022). However, in most industrial workflows this knowledge remains difficult to query systematically, forcing engineers to manually read reports and cross-check findings across documents.

Knowledge graphs offer a natural way to represent machine components, fault modes, and observed relations in a structured and machine-interpretable form. In the wind domain, multimodal knowledge graph approaches have already demonstrated value for explainable operations and maintenance decision support (Chatterjee & Dethlefs, 2021). More broadly, recent studies indicate that combining large language models (LLMs) with explicit knowledge representations can improve transparency and controllability compared with purely parametric QA systems (J. Z. Pan et al., 2023). This motivates ontology-grounded question answering pipelines in which natural language questions are translated into executable graph queries.

Despite this promise, robust text-to-SPARQL generation remains challenging. Recent LLM-based SPARQL systems report recurring structural and semantic errors, and therefore integrate retrieval support, schema guidance, and post-generation validation or repair steps (Allemang & Sequeda,

2024; Emonet, Bolleman, Duvaud, de Farias, & Sima, 2025; X. Pan, de Boer, & van Ossenbruggen, 2025; Arazzi, Ligari, Nicolazzo, & Nocera, 2025). For industrial inspection scenarios, these challenges are amplified by specialized terminology, sparse supervision, and the high practical cost of incorrect diagnostic answers.

To address this gap, we present an ontology-grounded question answering framework for wind turbine gearbox inspection reports. The framework converts inspection content into a domain ontology and knowledge graph, then uses an LLM to map user questions to SPARQL queries under ontology-based constraints. An Ontology-Based Query Checker (OBQC) (Allemang & Sequeda, 2024) validates generated queries against schema and semantic rules before execution, improving reliability for real diagnostic questions.

The main contributions of this paper are threefold: (i) a domain-grounded knowledge representation workflow for transforming unstructured gearbox inspection reports into queryable semantic data; (ii) a constrained LLM-based query generation pipeline with iterative ontology-based checking and repair; and (iii) an empirical evaluation on real-world inspection reports and prompts of varying complexity, showing high executable-query reliability in practical maintenance question answering.

2. RELATED WORK

Our work sits at the intersection of (1) natural language to SPARQL generation with large language models, (2) construction and use of ontologies and knowledge graphs for industrial maintenance knowledge, and (3) hybrid LLM–KG systems that add explicit constraints and checking to improve reliability.

2.1. LLM-based text-to-SPARQL generation

Recent work shows that large language models can translate natural language questions into SPARQL queries, but robustness remains a key issue for real deployments. A common observation is that models often generate queries that look plausible but contain schema mismatches, wrong identifiers, or invalid join patterns, especially for multi-hop questions or domain-specific schemas. D'Abramo et al. (2025) provide an extensive study of in-context learning setups for text-to-SPARQL and show that performance is sensitive to prompting choices and question complexity. Complementary evidence comes from studies that fine-tune LLMs for NL-to-SPARQL and analyze remaining failure modes; even after specialization, complex queries are still challenging and errors often relate to exact structural constraints of SPARQL and the underlying KG schema (Mecharnia & d'Aquin, 2025).

To address these issues, several systems add retrieval and guidance signals during generation. Emonet et al. (2025) propose an LLM-based pipeline for federated knowledge graphs that retrieves schema information and example queries and then applies a correction step, improving executability for real user questions. Pan et al. (2025) similarly argue that exposure to schema patterns and domain-specific query structures is crucial. They present a modular framework combining fine-tuned models and optional retrieval to improve SPARQL generation over scholarly knowledge graphs. These approaches motivate the use of schema-aware prompting and post-generation validation for executable querying, which becomes even more important in industrial settings where terminology is specialized and incorrect answers can be costly.

2.2. Ontologies and knowledge graphs for PdM and industrial maintenance text

In predictive maintenance, many decisions rely not only on sensor-derived anomalies but also on unstructured technical text such as inspection narratives, work orders, and maintenance actions. Prior work in the process industry shows that technical language can encode rich fault semantics and can be used as supervision or complementary evidence for intelligent fault diagnosis (Löwenmark et al., 2022). In the wind domain, Chatterjee and Dethlefs (2021) introduce a multimodal knowledge graph that integrates operational data and maintenance-related information, and demonstrate decision-support use cases based on query and reasoning. This supports the broader view that maintenance knowledge graphs can serve as a structured layer for searching, aggregation, and explainable analytics.

Building such graphs from industrial corpora remains difficult because ontologies require expert input and industrial text often contains nested entities, abbreviations, and site-specific terminology. Recent work explores using LLMs to assist with this step. van Cauter and Yakovets (2024) propose ontology-guided extraction from maintenance short texts using open LLMs and a small set of in-context examples, obtaining strong results without heavy fine-tuning. Other studies focus on fault-diagnosis knowledge bases and discuss practical challenges such as ontology subjectivity and data scarcity, while showing that LLM-assisted extraction can improve fine-grained knowledge graph construction from maintenance logs and technical documents (Liao et al., 2025). Beyond question answering, maintenance-oriented graphs have also been combined with graph learning for recommendation and planning, for example by constructing an ontology-based maintenance KG and applying graph neural models to suggest solutions and provide explanatory links (Xia, Liang, Leng, & Zheng, 2023).

2.3. Hybrid LLM–KG systems and ontology-based constraints for reliability

A central motivation for combining LLMs with KGs is to benefit from both flexible language understanding and explicit,

inspectable structure. Pan et al. (2023) summarize opportunities and challenges for such hybrid systems, highlighting grounding, controllability, and evaluation as recurring themes. Recent surveys on graph-based retrieval augmented generation further emphasize that graph structure can improve retrieval quality by selecting nodes, paths, or subgraphs that capture relations needed for multi-hop questions, instead of treating the knowledge base as flat text (Peng et al., 2024; Chen, 2025).

For executable querying, explicit constraints are particularly valuable. Allemang and Sequeda (2024) propose ontology-based query checking where an LLM-generated SPARQL query is validated against ontology semantics, and violations are used to guide correction. This line of work aligns with the broader trend of adding verification and repair steps to reduce hallucinations and schema violations in text-to-query pipelines. In industrial maintenance QA, such safeguards are important because domain ontologies encode allowable component relations and fault semantics, and these constraints can be used to prevent invalid or misleading query execution.

Overall, prior work establishes that LLM-based text-to-SPARQL is feasible but error-prone, that industrial maintenance benefits from structuring technical text into ontologies and knowledge graphs, and that ontology-aware checking and graph-structured retrieval are promising mechanisms for improving reliability. Our approach builds on these insights by combining example-based SPARQL generation with ontology-grounded validation and iterative repair for querying wind turbine inspection knowledge.

3. METHODOLOGY

We present a Retrieval-Augmented Generation (RAG) system for querying RDF ontologies using natural language. The system comprises four main components: ontology construction from domain documents, ontology-based knowledge representation, query generation via large language models, and ontology-based query checking. The methodology is designed for the wind turbine gearbox inspection domain but generalises to other ontology-backed knowledge bases.

For generation, we run the `CodeLlama-13B` model through Ollama in fully local mode rather than external hosted APIs. This choice prioritizes confidentiality of inspection data and prompts, keeps model behavior under local control (versioning and deployment), and improves reproducibility in restricted or offline industrial environments.

3.1. Input Data

The input to our pipeline is a collection of inspection reports converted to structured markdown. Each report follows a consistent layout. First, it includes a hierarchical table describing the gearbox structure in terms of stages and components. Second, it provides report-level metadata such as inspection date, turbine information, and operational data.

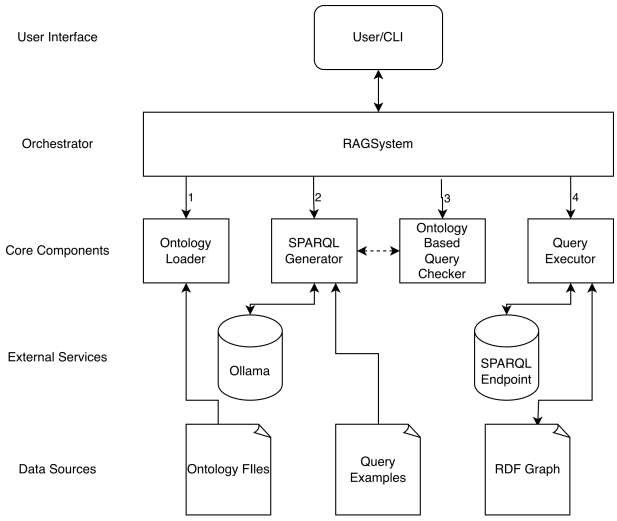

Figure 1. Overview of the proposed RAG pipeline for ontology-backed natural language querying.

The core body then contains a sequence of finding tables. Each finding table captures the key fields used in downstream graph construction and querying: severity, finding type, textual description, picture reference, stage, and component abbreviation. This tabular representation is followed by free-text conclusion sections that summarize overall inspection outcomes and recommendations.

3.2. Ontology Construction

The knowledge base is generated automatically from markdown inspection reports (generated from PDFs using the `Docling` Python library) (Team, 2024). A parser first extracts report metadata and finding tables into a structured JSON representation. An ontology generator then consumes this JSON to infer gearbox stages and components, create turbine and report instances, and write RDF findings with provenance links back to the source documents. Only the core schema and fault-type taxonomy are hand-authored; all inspection-specific instance data is produced by the pipeline and can be fully regenerated when the source reports change. The ontology is explicitly designed to be *LLM-friendly*—that is, structured to facilitate reliable SPARQL query generation by language models.

3.2.1. Schema Design

The schema defines classes, properties, and SHACL shapes. We prioritise clarity for both humans and language models so that query intent maps cleanly to ontology terms. The main design choices are:

- **Simple filtering paths:** Frequently used identifiers (turbine, stage, and component abbreviations) are exposed in

a form that supports direct and readable query filters.

- **Clear role separation:** We distinguish properties used for structural modelling from those intended for retrieval, reducing ambiguity during query generation.
- **Deterministic ranking:** Severity is represented with an explicit numeric order so sorted outputs are stable and interpretable.
- **Interoperable semantics:** Established vocabularies (SOSA, PROV-O, DCTERMS, SKOS) are reused to keep the ontology compatible with common RDF tooling and practices.
- **Constrained usage:** Domain and range constraints guide valid property use and help both validation and automated query checking.

3.2.2. Hierarchical Component Structure Extraction

We first construct the *ontology* as a stable description of gearbox knowledge, independent of any single inspection event. From the parsed JSON records, the generator automatically extracts recurring structural concepts—gearbox stages, component types, and their relationships—and organises them into a hierarchical component model. In practice, this means consolidating naming variants (for example, different abbreviations that refer to the same stage) and preserving a consistent stage-to-component hierarchy.

This ontology-first design is intentional: by fixing the conceptual structure before loading observations, we make query generation more robust and easier to control. The resulting file captures the canonical gearbox structure together with shared domain vocabulary (including the manually curated fault taxonomy), ensuring downstream queries rely on a single coherent schema while the extracted structural instances remain reproducible from the reports.

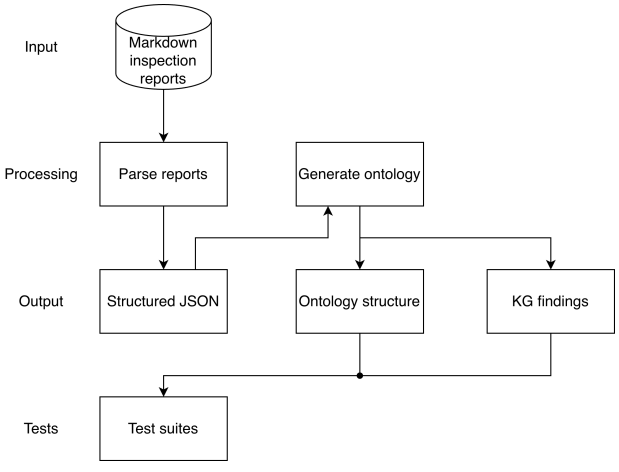

Figure 2. Ontology construction workflow from inspection reports to validated ontology and findings graph artifacts.

3.3. Findings Knowledge Graph Construction

After the ontology is established, the same automated pipeline populates a separate *knowledge graph* with inspection-specific evidence. This graph stores turbines, reports, and individual findings, while linking each finding back to ontology concepts such as stage and component type. Keeping findings separate from the structural ontology allows us to reuse the same conceptual backbone across turbines and reporting periods without redefining domain structure.

Each finding is tied to its source report and annotated with provenance so that query results remain auditable and traceable to the original inspection context. Because the instance graph is generated rather than manually edited, updates to markdown reports can be reflected by rerunning the parser and generator instead of hand-maintaining RDF triples.

3.4. Validation

Validation acts as a design guardrail rather than a final clean-up step. We use SHACL to enforce a small set of mandatory attributes and links for each finding (identity, location in the gearbox hierarchy, and severity), because these are the fields most frequently used in retrieval and filtering. This avoids silent schema drift when new reports are added and keeps query behaviour predictable.

Severity values are constrained to a controlled set so ranking remains stable across reports, and RDFS-aware checking ensures that constraints still hold when subclass relations are involved. We prioritise strictness on high-impact fields and flexibility elsewhere to balance data quality with practical ingestion.

3.5. Ontology-based Knowledge Representation

The RAG system represents knowledge as RDF graphs loaded from Turtle files. We separate the canonical ontology structure from inspection findings so that conceptual changes and data updates can evolve independently. This improves maintainability: structural concepts only have to be ingested once, while findings are updated continuously.

3.5.1. Schema Extraction

Before query generation, we build a compact schema string from the ontology T-box (the terminology layer), not from full report content. In practice, the ontology is loaded from Turtle files and can run in a dual-ontology mode: one file defines structural concepts (for example turbines, stages, and components) and another defines findings. These graphs are merged for querying, and schema extraction is performed over this merged RDFS view.

The extracted schema includes: (i) namespace prefixes, (ii) classes, and (iii) properties with their domain and range.

We collect this using standard RDFS signals (class declarations and subclass links, plus property declarations with domain/range). In the prompt, properties are grouped by domain class and shown with their range so the model can tell whether a property expects a literal value or another entity. This simple representation helps the model choose valid predicates and avoid type-mismatch errors.

We may also include a small capped set of representative instances to illustrate data shape, but this is auxiliary context. The primary prompt context remains schema-level because it is more stable and less noisy than raw instance triples.

3.5.2. Domain Taxonomy

The taxonomy text used in prompting is a controlled two-level hierarchy: stages at the parent level and their components at the child level, each with official abbreviations and full names. Abbreviation formatting is normalized to match IRI conventions used in the knowledge base (for example, whitespace normalization), so the model can produce valid identifiers in filters and patterns.

To keep context focused and avoid exposing inspection wording, finding-type tables are not injected as taxonomy text. Severity is instead handled through fixed prompting rules. In short, taxonomy provides controlled vocabulary for stage/component identifiers without adding unnecessary report detail.

3.5.3. Query Backend

The backend supports both local graphs and remote SPARQL endpoints. This dual mode is a deliberate trade-off: local execution simplifies development and debugging, while endpoint execution supports larger datasets. In both cases, schema extraction stays local so prompt construction remains stable across deployment environments.

3.6. Query Generation

Query generation combines few-shot prompting with retrieval of semantically similar examples. The method constrains generation through context instead of relying on unconstrained free-form synthesis, which improves reliability for structured queries.

In our local deployment, the combined query generation and execution cycle takes about 15 seconds on average per user question.

3.6.1. Example Selection

For each user question, we retrieve the most similar examples from a curated question–SPARQL pool containing 106 few-shot examples. Semantic similarity is computed using the `all-MiniLM-L6-v2` embedding model from `sentence-transformers`. Dynamic selection is preferred over fixed examples because query intent varies: structural questions and finding-centric questions require different triple patterns. Matching by semantic similarity gives the model targeted patterns at inference time without increasing manual prompt engineering.

3.6.2. Prompt Construction

The prompt is intentionally layered. In the initial generation call, we concatenate a system block with fixed SPARQL rules (class/property usage, variable binding, query structure, and component-filtering constraints) and top-K retrieved question–query examples, then append a user block with the current question, extracted ontology schema, and optional taxonomy. This order reflects dependency: the model first internalizes constraints, then observes valid patterns, then applies both to the request.

If ontology-based checking fails, we issue a shorter repair prompt rather than replaying the full template. The repair call includes the original question, schema (and taxonomy when available), the validation error list, the incorrect query, and a small set of relevant examples (up to three), followed by targeted fix instructions and a requirement to return only corrected SPARQL. We use low-temperature decoding in both stages to reduce syntactic variance and favor reproducible query forms.

3.6.3. Post-Processing

Post-processing performs deterministic repairs for common, low-level mistakes (such as missing prefixes, malformed aggregates, or unbound filter variables). We isolate these corrections from the LLM so predictable syntax issues are handled by rules, reserving model capacity for semantic mapping. This hybrid design reduces avoidable execution failures and improves repeatability.

3.7. Query Checking

Before execution, an Ontology-Based Query Checker (OBQC) (Allemang & Sequeda, 2024) validates semantic compatibility with the ontology. The checker uses schema-level knowledge only, so validation stays robust even when instance data is sparse or evolving.

3.7.1. Syntax and BGP Extraction

The checker first confirms parseable SPARQL, then extracts core triple patterns to obtain a normalised representation of the query. This simplification step makes downstream rule checks more transparent and less sensitive to stylistic differences in how equivalent queries are written.

3.7.2. Semantic Rules

The semantic rules focus on failure modes that most often break execution or return misleading results:

- **Domain and range compatibility:** Subjects and objects must match the expected class constraints of each property.
- **Property-chain coherence:** Consecutive triples must connect classes that are semantically compatible.
- **Vocabulary validity:** Predicates must come from the ontology (or standard RDF namespaces).
- **Variable and literal correctness:** Variables must be bound before use, and literals cannot replace required entity nodes.

3.7.3. Repair Mechanism

If a query fails validation, the system attempts targeted repair by feeding the model the original question, schema context, the invalid query, and explicit error messages. This keeps correction grounded in both user intent and ontology constraints. We cap repair attempts to avoid unproductive loops; unresolved queries are returned as "unknown" rather than executed unsafely.

4. EVALUATION

We evaluate the proposed system in an end-to-end setting: a natural-language question is converted into SPARQL, executed on the target endpoint, and compared against gold answers. The goal is to measure practical question-answering correctness rather than only query string similarity.

4.1. Experimental Setup

The evaluation uses a held-out gold dataset with 50 manually curated questions. The dataset is independent of few-shot/training examples and built from an evaluation question pool where each question is paired with a manually written, verified gold SPARQL query. This independence ensures that the evaluation examples are not reused from the prompting context, making query-text metrics such as BLEU and ROUGE more meaningful as measures of generalization rather than memorized lexical overlap. Each example contains a natural-language question, gold SPARQL, and a gold answer set obtained by executing the gold query. Examples with invalid or non-executable gold queries are excluded during dataset construction.

For each question, the full inference pipeline is executed end-to-end: natural-language question, SPARQL generation, optional OBQC/repair and query execution against the same endpoint used for gold answers. Ontology-based query checking (OBQC) and automatic repair are enabled in the reported run. This makes the evaluation operationally realistic, since query validity constraints and execution behavior are part of the measured performance.

4.2. Metrics

We use three groups of metrics: answer-level quality, query success rates, and query-text similarity.

Answer-level quality. Mean Precision, Mean Recall, and Mean F1 are based on comparing the generated answer set with the gold answer set for each question. Before comparison, rows are normalized so equivalent field names are treated consistently. A generated row is matched to the best gold row using overlap in field–value pairs, with semantic fallback matching when direct field overlap is missing. Precision is the fraction of generated rows that are correct, recall is the fraction of gold rows that are recovered, and F1 balances both. We compute these per question and then average across all questions.

Exact Match Rate is stricter. A question counts as an exact match only when the generated and gold answers are fully identical after normalization: same number of rows and exactly the same row content. The final rate is the fraction of questions that meet this condition.

Query success rates. These metrics track whether the system successfully completes key pipeline steps. Query Generation Rate is the fraction of questions for which the system produces a SPARQL query. Executability Rate is the fraction for which the generated query executes without runtime error. OBQC Trigger Rate is the fraction of questions whose initial generated query fails validation and therefore requires the OBQC repair path. Repair Success Rate is computed only over triggered cases and measures the fraction of OBQC-triggered queries that are successfully repaired before execution or final evaluation.

Query-text similarity. These metrics compare the generated SPARQL text with the gold SPARQL text directly.

Mean BLEU measures n-gram precision: it checks how many short token sequences in the generated query also appear in the gold query. In practice, this captures local phrasing and operator/token choices. The score uses tokenized lowercase text and smoothing so short queries are not unfairly penalized when higher-order n-grams are sparse. We compute BLEU per question and report the mean.

Mean ROUGE-1, ROUGE-2, and ROUGE-L measure overlap from a recall-oriented perspective and we report their F1 forms. ROUGE-1 uses unigram overlap (single tokens), ROUGE-2 uses bigram overlap (token pairs), and ROUGE-L uses longest common subsequence overlap, which captures global ordering

similarity even when tokens are not contiguous. Together, these ROUGE variants reflect lexical overlap, short-pattern overlap, and overall structural similarity. Each is computed per question and then averaged.

4.3. Main Results

Table 1 summarizes the results from the main evaluation run.

Table 1. End-to-end evaluation results on the gold dataset (50 questions).

Metric	Value
Mean Precision	0.9600
Mean Recall	0.9413
Mean F1	0.9425
Exact Match Rate	0.3800
Query Generation Rate	1.0000
Executability Rate	0.9600
OBQC Trigger Rate	0.1000
Repair Success Rate (when triggered)	0.6000
Mean BLEU	0.8349
Mean ROUGE-1	0.9406
Mean ROUGE-2	0.9031
Mean ROUGE-L	0.9227

The system achieves strong answer-level performance (mean F1 = 0.9425) while maintaining high generation and execution reliability in this benchmark. OBQC was triggered for 5 out of 50 queries (10.0%), with 3 out of 5 triggered cases repaired successfully (60.0%) and 2 out of 5 failing or exhausting the repair path. The exact match rate is substantially lower (38%) because exact match requires full answer-set identity, whereas F1 rewards overlap and partial row matches. Row order does not affect exact match, and column names are partially normalized before comparison. Nevertheless, extra or missing bindings can sharply reduce exact match even when most returned answers are correct. Thus, the gap between exact match and mean F1 primarily suggests minor binding-level discrepancies rather than severe semantic failure.

4.4. Stratified Results by Complexity

Table 2 reports performance by question complexity. To keep the comparison focused, the table includes only the most diagnostic answer-level metrics: mean F1 and exact match.

Table 2. Evaluation results stratified by question complexity.

Complexity	n	Mean F1	Exact Match
Simple	6	1.0000	0.6667
Medium	29	0.8966	0.2414
Complex	15	0.8911	0.3333

The simple subset is solved completely at the answer level, while medium and complex questions show lower F1 and exact-match rates. This pattern indicates that the remaining errors concentrate in queries requiring more elaborate binding structure, such as multi-step ontology paths, field selection, or aggregation semantics. The lower exact-match rates in the medium and complex subsets are consistent with the aggregate results: many outputs preserve substantial answer overlap but fail to reproduce the full gold answer set exactly.

4.5. Qualitative Success and Failure Examples

To complement aggregate metrics, we present two successful exact-match cases (Examples 1 and 2), followed by one representative failure case (Example 3).

Example 1 (Success): Simple retrieval. Question: "Retrieve all wind turbine identifiers in the knowledge base."

Gold query:

```
PREFIX wt: <https://example.org/wt#>

SELECT ?turbineId WHERE {
    ?turbine a wt:WindTurbine .
    ?turbine rdfs:label ?turbineId .
}
```

Generated query:

```
PREFIX wt: <https://example.org/wt#>
SELECT ?turbineId
WHERE {
  ?turbine a wt:WindTurbine .
  ?turbine rdfs:label ?turbineId .
}
```

The generated query is semantically equivalent to the gold query: it uses the same class (`wt:WindTurbine`), the same identifier property (`rdfs:label`), and the same output variable (`?turbineId`).

Generated results: identical 15 bindings for `?turbineId`.

Outcome: Precision = 1.0, Recall = 1.0, F1 = 1.0, Exact Match = true; query executable and OBQC passed.

Example 2 (Success): Aggregation with filter. Question: "Count the total number of findings with severity level 'Severe' across all turbines."

Gold query:

```
PREFIX wt: <https://example.org/wt#>
SELECT (COUNT(?finding) AS ?count)
WHERE {
    ?finding a wt:Finding .
    ?finding wt:severity wt:Severe .
}
```

Generated query:

```
PREFIX wt: <https://example.org/wt#>
SELECT (COUNT(?finding) AS ?count)
```

```
WHERE {
  ?finding a wt:Finding .
  ?finding wt:severity wt:Severe .
}
```

The generated query is structurally identical in intent: same target class (`wt:Finding`), same ontology-value filter (`wt:severity wt:Severe`), and same aggregation expression (`COUNT(?finding) AS ?count`).

Outcome: Precision = 1.0, Recall = 1.0, F1 = 1.0, Exact Match = true; BLEU = 1.0, ROUGE-1/2/L = 1.0; query executable and OBQC passed.

Example 3 (Failure): Missing `GROUP BY` (aggregation semantics). Question: "How many findings does each turbine have? Return turbine identifier and count."

Gold query:

```
PREFIX wt: <https://example.org/wt#>
SELECT ?turbineId
       (COUNT(?finding) AS ?count)
WHERE {
    ?finding a wt:Finding .
    ?finding wt:turbineId ?turbineId .
}
GROUP BY ?turbineId
```

Generated query:

```
PREFIX wt: <https://example.org/wt#>
SELECT ?turbineId
       (COUNT(?finding) AS ?count)
WHERE {
  ?finding a wt:Finding .
  ?finding wt:turbineId ?turbineId .
}
```

The generated query captures the right class, relation, and aggregation function, but it omits `GROUP BY ?turbineId`. Without this clause, SPARQL aggregates over the full solution set, producing one global count instead of one count per turbine.

Gold results: 15 rows (one per turbine).
Generated results: 1 row.

Outcome: Precision = 1.0, Recall = 0.067, F1 = 0.125, Exact Match = false; query executable and OBQC passed. Graph-level structural metrics still match strongly; the key error is missing grouping semantics. This is a semantic failure, not a syntax or ontology failure. The model captures "count findings" but misses that "each turbine" must be realized as `GROUP BY ?turbineId`.

Example 4 (OBQC Repair): Unbound variable in `FILTER`. Question: "Find findings for turbines WTA01, WTG07, or WTD05."

Original query (incorrect):

```
PREFIX wt: <https://example.org/wt#>
SELECT ?finding ?faultType ?severity
       ?componentAbbreviation
WHERE {
  ?finding a wt:Finding .
  FILTER (
    ?aboutTurbine = "WTA01" ||
    ?aboutTurbine = "WTG07" ||
    ?aboutTurbine = "WTD05"
  )
  ?finding wt:faultType ?faultType .
  ?finding wt:severity ?severity .
  ?finding wt:componentAbbreviation
           ?componentAbbreviation .
}
```

OBQC error: "The variable `?aboutTurbine` is used but never bound. Make sure all variables appear in at least one triple pattern (as subject, predicate, or object), or are bound through `BIND`/`VALUES` clauses."

Repaired query:

```
PREFIX wt: <https://example.org/wt#>
SELECT ?finding ?faultType ?severity
       ?componentAbbreviation
WHERE {
  ?finding a wt:Finding .
  ?finding wt:turbineId ?turbineId .
  ?finding wt:faultType ?faultType .
  ?finding wt:severity ?severity .
  ?finding wt:componentAbbreviation
           ?componentAbbreviation .
  FILTER (
    ?turbineId = "WTA01" ||
    ?turbineId = "WTG07" ||
    ?turbineId = "WTD05"
  )
}
```

The error does not come from the filter values. The issue is that the query filters on `?aboutTurbine`, but this variable is never linked to any triple pattern. The repair fixes this by adding `?finding wt:turbineId ?turbineId .` and then using `?turbineId` in the filter. This keeps the original intent (only turbines WTA01, WTG07, and WTD05) and makes the query valid. More importantly, it shows that the OBQC can fix semantic mistakes: query can look syntactically correct but still be logically incomplete. After repair, the constraint is explicit, so the query is easier to trust and maintain.

4.6. Error Analysis

Failures are defined as cases with F1 < 1.0 (answer mismatch), while errors for taxonomy include failures and execution/validation issues categorized by query-level cause. In this run, 6 out of 50 examples are failures (F1 < 1.0), with failures concentrated in the medium and complex subsets.

The failures are semantically meaningful and include cases such as stage/component-type questions that return zero rows due to an incorrect ontology path pattern and aggregation queries that omit required grouping, yielding global aggregates instead of per-entity counts. This pattern suggests that the current bottleneck is not basic query generation, but semantic query structure (path construction, aggregation, and field selection) in specific edge cases.

5. DISCUSSION

The updated results confirm that the proposed pipeline is practical for ontology-grounded question answering at larger evaluation scale. Across all 50 examples, the system generates a query for every question and executes successfully in 96% of cases. OBQC is triggered for 5 queries (10%); 3 of these are repaired successfully, giving a repair success rate of 60% over triggered cases.

Exact match requires identical columns and row sets after normalization, whereas answer-level F1 measures returned factual correctness under semantic row matching. Consequently, many non-exact predictions still recover most or all relevant answers. The 38% exact-match rate should therefore be interpreted together with the much higher mean F1 of 0.9425: it indicates frequent minor binding- or answer-set discrepancies, not widespread failure.

The observed failure cases are informative. In total, 6 out of 50 examples have F1 below 1.0, concentrated in the medium and complex subsets. Representative errors include ontology-path mistakes in stage/component-type retrieval that return empty result sets and an aggregation query that uses `COUNT` without `GROUP BY ?turbineId`, collapsing per-entity results into one aggregate row. These are genuine semantic/query-structure errors rather than simple syntax artefacts.

From an engineering perspective, the priority is therefore improving semantic control while also reducing the remaining execution and repair failures. Promising directions include stronger ontology-path grounding, aggregation-aware decoding constraints, more targeted OBQC repair prompts, and answer-level reranking among multiple candidate queries.

6. CONCLUSION

This paper presented an end-to-end natural-language-to-SPARQL framework that combines LLM-based query generation with ontology-based query checking and automatic repair. On the latest gold-dataset evaluation (50 questions, OBQC enabled), the system achieves strong answer-level quality (mean F1 = 0.9425, precision = 0.9600, recall = 0.9413) with high operational reliability: 100% query generation, 96% executability, a 10% OBQC trigger rate, and 60% repair success over triggered cases.

The remaining failures are primarily semantic rather than syntactic. Across the evaluation, 6 out of 50 examples have F1 below 1.0, including wrong ontology-path selection in stage/component-type questions and missing aggregation structure (`GROUP BY`) in a per-turbine count query. This indicates that the core pipeline is robust, but that next improvements should target semantic query planning, field selection, aggregation decisions, and repair coverage for the small number of validation-triggered cases.

Future work will extend evaluation to larger and more diverse benchmarks, including cross-ontology and cross-endpoint settings, and investigate methods such as ontology-guided candidate generation, aggregation-aware constraints, answer-level reranking, and approaches that combine structured and unstructured data sources (e.g., integrating anomaly-detection records with textual evidence) to reduce residual semantic errors while preserving reliability.

Overall, the results support reliability-oriented design as a practical path toward trustworthy NL interfaces for knowledge-graph querying.

REFERENCES

Achouch, M., Dimitrova, M., Ziane, K., Sattarpanah Karganroudi, S., Dhouib, R., Ibrahim, H., & Adda, M. (2022, January). On Predictive Maintenance in Industry 4.0: Overview, Models, and Challenges. *Applied Sciences*, *12*(16), 8081. doi: 10.3390/app12168081

Allemang, D., & Sequeda, J. (2024, May). *Increasing the LLM Accuracy for Question Answering: Ontologies to the Rescue!* (No. arXiv:2405.11706). arXiv. doi: 10.48550/arXiv.2405.11706

Arazzi, M., Ligari, D., Nicolazzo, S., & Nocera, A. (2025, February). *Augmented Knowledge Graph Querying leveraging LLMs* (No. arXiv:2502.01298). arXiv. doi: 10.48550/arXiv.2502.01298

Chatterjee, J., & Dethlefs, N. (2021, February). *XAI4Wind: A Multimodal Knowledge Graph Database for Explainable Decision Support in Operations & Maintenance of Wind Turbines* (No. arXiv:2012.10489). arXiv. doi: 10.48550/arXiv.2012.10489

Chen, R. (2025, March). Retrieval-Augmented Generation with Knowledge Graphs: A Survey. In *Computer Science Undergradaute Conference 2025 @ XJTU*.

D'Abramo, J., Zugarini, A., & Torroni, P. (2025, May). Investigating Large Language Models for Text-to-SPARQL

Generation. In W. Shi et al. (Eds.), *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing* (pp. 66–80). Albuquerque, New Mexico, USA: Association for Computational Linguistics. doi: 10.18653/v1/2025.knowledgenlp-1.5

Emonet, V., Bolleman, J., Duvaud, S., de Farias, T. M., & Sima, A. C. (2025, February). *LLM-based SPARQL Query Generation from Natural Language over Federated Knowledge Graphs* (No. arXiv:2410.06062). arXiv. doi: 10.48550/arXiv.2410.06062

Liao, X., Chen, C., Wang, Z., Liu, Y., Wang, T., & Cheng, L. (2025, May). Large language model assisted fine-grained knowledge graph construction for robotic fault diagnosis. *Advanced Engineering Informatics*, *65*, 103134. doi: 10.1016/j.aei.2025.103134

Löwenmark, K., Taal, C., Schnabel, S., Liwicki, M., & Sandin, F. (2022, October). Technical Language Supervision for Intelligent Fault Diagnosis in Process Industry. *International Journal of Prognostics and Health Management*, *13*(2). doi: 10.36001/ijphm.2022.v13i2.3137

Mecharnia, T., & d'Aquin, M. (2025, January). Performance and Limitations of Fine-Tuned LLMs in SPARQL Query Generation. In G. A. Gesese, H. Sack, H. Paulheim, A. Merono-Penuela, & L. Chen (Eds.), *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)* (pp. 69–77). Abu Dhabi, UAE: International Committee on Computational Linguistics.

Pan, J. Z., Razniewski, S., Kalo, J.-C., Singhania, S., Chen, J., Dietze, S., … Graux, D. (2023, August). *Large Language Models and Knowledge Graphs: Opportunities and Challenges* (No. arXiv:2308.06374). arXiv. doi: 10.48550/arXiv.2308.06374

Pan, X., de Boer, V., & van Ossenbruggen, J. (2025, August). *FIRESPARQL: A LLM-based Framework for SPARQL Query Generation over Scholarly Knowledge Graphs* (No. arXiv:2508.10467). arXiv. doi: 10.48550/arXiv.2508.10467

Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., … Tang, S. (2024, September). *Graph Retrieval-Augmented Generation: A Survey* (No. arXiv:2408.08921). arXiv. doi: 10.48550/arXiv.2408.08921

Team, D. S. (2024, 8). *Docling technical report* (Tech. Rep.). Retrieved from `https://arxiv.org/abs/2408.09869` doi: 10.48550/arXiv.2408.09869

van Cauter, Z., & Yakovets, N. (2024, August). Ontology-guided Knowledge Graph Construction from Maintenance Short Texts. In R. Biswas, L.-A. Kaffee, O. Agarwal, P. Minervini, S. Singh, & G. de Melo (Eds.), *Proceedings of the 1st Workshop on Knowledge Graphs and Large Language Models (KaLLM 2024)* (pp. 75–84). Bangkok, Thailand: Association for Computational Linguistics. doi: 10.18653/v1/2024.kallm-1.8

Xia, L., Liang, Y., Leng, J., & Zheng, P. (2023, April). Maintenance planning recommendation of complex industrial equipment based on knowledge graph and graph neural network. *Reliability Engineering & System Safety*, *232*, 109068. doi: 10.1016/j.ress.2022.109068