University of Kent

Kent Academic Repository

Karkee, Shikhat, Botoeva, Elena, Coombes, Sam, Jordanous, Anna, Kafalı, Özgur and Lanti, Davide (2025) *Accessing Semi-structured Data with RML and LLMs*. In: Proceedings of the 38th International Workshop on Description Logics - DL 2025. CEUR Workshop Proceedings . CEUR

Downloaded from
https://kar.kent.ac.uk/114635/ The University of Kent's Academic Repository KAR

The version of record is available from
https://ceur-ws.org/Vol-4091/paper54.pdf

This document version
Author's Accepted Manuscript

DOI for this version

Licence for this version
CC BY (Attribution)

Additional information

Versions of research works

Versions of Record
If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts
If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in ***Title of Journal*** , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries
If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies).

Accessing Semi-structured Data with RML and LLMs

Shikhat Karkee[1], Elena Botoeva[1], Sam Coombes[2], Anna Jordanous[1], Özgür Kafali[1] and Davide Lanti[3]

[1]University of Kent, Canterbury, UK
[2]East Kent Hospitals University Foundation Trust, Ashford, UK
[3]Free University of Bozen-Bolzano, Bolzano, Italy

Abstract

We demonstrate a principled way to query semistructured data by integrating natural language processing capabilities into knowledge graph generation. We instantiate RML mappings with LLM queries to extract ontological terms from plain text. We evaluate our approach on pharmaceutical data in JSON format.

Keywords

Semi-structured data, Large Language Models, Ontology-Based Data Access, RML

1. Introduction

The Ontology-Based Data Access (OBDA) framework [1] has provided the theoretical foundations for representing data stored in a database as a *knowledge graph* – a collection of facts formulated using high-level concepts and relationships. A standard formalism for describing knowledge graphs in practice is the Resource Description Framework (RDF) [2]. Once the data has been represented as a knowledge graph, it can be analysed using a formal query language with a well-defined semantics, for instance such as SPARQL, a standard query language for RDF [3]. Importantly, for such *structured* data representations and query languages, there is a guarantee that the answers are correct as long as the data is correct.

A crucial component of the OBDA framework is *mappings*, which declaratively specify how knowledge graph entities should be populated from values found in the data. Historically the prevailing database management systems were relational, so originally OBDA was used to facilitate access to relational data and the mapping languages were only concerned with relational data sources. R2RML, the RDB to RDF Mapping Language, was developed as a W3C recommendation for specifying declarative mappings from relational databases to RDF datasets [4]. With the advent of non-relational databases, the OBDA framework has been extended to accommodate alternative data models [5, 6]. Likewise, in line with the general trend of data in non-relational formats being widely available on the Web, an extension of R2RML to support generic data sources has been under development, resulting in RML [7, 8], the RDF Mapping Language. RMLMapper is a library for generating knowledge graphs from various data formats, including CSV, JSON, and XML [9].

Despite the recent advances in accommodating a wider range of data formats, a fundamental assumption when generating knowledge graphs following the OBDA principle is that the data are inherently structured. However, a significant portion of data available on the Internet is, at best, semi-structured. As a running example, consider online databases that catalogue medicines and information about them [10]. Notably, information such as the name of a drug or its interactions with other drugs is structured. Other bits of information, though, may come as unstructured text. For instance, "1 g, every 4–6 hours; maximum 4 g per day" is a dosage information for paracetamol. In general, dosage instructions may come in a variety of formats, even for the same drug, e.g., "1 g, every 4–6 hours, dose

DL 2025: 38th International Workshop on Description Logics, September 3–6, 2025, Opole, Poland
sk992@kent.ac.uk (S. Karkee); e.botoeva@kent.ac.uk (E. Botoeva); sam.coombes@nhs.net (S. Coombes); a.k.jordanous@kent.ac.uk (A. Jordanous); r.o.kafali@kent.ac.uk (Ö. Kafali); dalanti@unibz.it (D. Lanti)
0000-0001-5881-0258 (E. Botoeva); 0000-0003-2076-8642 (A. Jordanous); 0000-0001-9296-2087 (Ö. Kafali); 0000-0003-1097-2965 (D. Lanti)
© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

to be administered over 15 minutes; maximum 4 g per day". Assuming a scenario where the goal is to extract a knowledge graph from such a database in order to provide answers to pharmaceutical queries, the dosage information would be expected to have a structured representation. This highlights the necessity for extracting structured information from unstructured data.

The standard solution when dealing with unstructured data nowadays is to employ large language models (LLMs)–natural language processing tools that excel at understanding and generating language [11]. For instance, a state-of-the-art LLM would be capable of extracting the dosage amount and recommended administration frequency from a dosage instruction string as above. In the context of knowledge graph construction from text corpora, LLMs have been used for named entity recognition, relation extraction, as well as for fact generation [12, 13, 14, 15, 16, 17]. Despite the advances, however, the main challenge remains in terms of factual correctness of the extracted assertions, due to the so-called *hallucinations* [18], the consequence of LLMs being essentially statistical models outputting words that have high probability of continuing the current phrase. A pure LLM-based solution to knowledge graph construction, therefore, may not be acceptable in a high stakes application, where either the generated knowledge graph would be expected to be factually correct, or the correctness of query answers would need to be easily verifiable.

To combine the best of two worlds, the ability of LLMs to intelligently process arbitrary natural language and the correctness guarantees associated with the OBDA approach, we propose leveraging LLMs in a more surgical fashion. Namely our methodology involves integrating LLM queries within the RML mappings, where the input to the LLM is computed by the source part of a mapping rule and the output of the LLM is used to generate RDF terms. There are several advantages of our approach over an end-to-end LLM solution, where pharmaceutical queries are answered by an LLM directly over the data sources. First, the LLM is not used to deal with information that is already structured and can be processed using the standard RML rules. Second, the LLM is fed only short and focused snippets of text, thus reducing the risk of confusing it. Finally, it allows for a more straightforward verification of the correctness of the knowledge graph/query answers, as each fact generated by an LLM could be annotated with the text used to extract it. To the best of our knowledge, this is the first such application of LLMs in the OBDA context. The only other work that used LLMs within the OBDA framework employed LLMs to generate mappings automatically [19]. Other information extraction techniques have been used to convert textual documents to structured representations [20, 21], however these techniques are less powerful than LLMs.

We instantiate and evaluate our approach on a Medicines Information (MI) use-case. We use a small JSON dataset created to mimic the existing online MI datasets and a number of SPARQL queries. We compare our approach to the pure LLM-based solution and to the pure RML-based solution. We assess the correctness and completeness of the resulting knowledge graphs and of the query answers. Our experiments show that integrating LLMs into RML results in fewer hallucinations than the pure LLM solution, and as expected, makes more information available for structured processing comparing to the pure RML solution.

2. Preliminaries

Knowledge Graphs. A *Knowledge Graph* (KG) is a structured representation of information in the form of a graph, where nodes represent entities of interest, and edges represent the semantic relationships between those entities. The Resource Description Framework (RDF) is a W3C-standard model for data interchange on the Web and serves as the foundational representation format for specifying Knowledge Graphs (KGs). In RDF, knowledge is encoded as a set of triples of the form $(s\ p\ o)$, where s denotes the *subject*, p the *predicate*, and o the *object*. The subject s is either an Internationalized Resource Identifier (IRI) or a *blank node*[1], the predicate p is an IRI specifying the relationship, and the object o is an IRI, a blank node, or a *literal*[2]. For example, the triple `:aspirin :usage :indication1` expresses that

[1] An unnamed resource

[2] A concrete data value such as a string, number, or date

the drug aspirin has usage for an indication identified by the `:indication1` IRI, whereas the triple `:dosage1 :amount "100"^^xsd:decimal` states that the amount for the dosage entity `:dosage1` is 100, where `"100"^^xsd:decimal` is an RDF literal with an associated datatype. An *RDF graph* is a finite set of RDF triples. SPARQL is a standard query language for RDF [3].

Knowledge graph extraction from data. In traditional OBDA, one provides access to an (external) relational database (DB) through an ontology, which is connected to the DB by means of *mappings*. Given a DB schema $\mathcal{S}$ and an ontology $\mathcal{T}$, a (GAV) *mapping assertion* between $\mathcal{S}$ and $\mathcal{T}$ is an expression of the form $q(\vec{x}) \rightsquigarrow \tau(\vec{x})$ where $q(\vec{x})$ is a (SQL) query over $\mathcal{S}$, also called the *source* query, and $\tau(\vec{x})$ an RDF triple template over $\mathcal{T}$, also called the *target* query. A triple template $(f(\vec{x})\ \texttt{rdf:type}\ A)$, for a class name A in $\mathcal{T}$ and a template function f, asserts that the IRI computed as $f(\vec{x})$ is an instance of A, whereas $(f(\vec{x})\ P\ f'(\vec{x}))$, for a property name P in $\mathcal{T}$, asserts that $f(\vec{x})$ is connected to $f'(\vec{x})$ via P. Note that template functions are used to build IRIs, literals, or blank-nodes in an RDF triple.

Given a database instance D of $\mathcal{S}$ and a set $\mathcal{M}$ of mapping assertions between $\mathcal{S}$ and $\mathcal{T}$, we denote by $\mathcal{M}(D)$ the *KG generated by $\mathcal{M}$ from D* (which is an RDF graph):

$$\{(f(\vec{o})\ \texttt{rdf:type}\ A) \mid \vec{o} \in ans(\psi, D) \text{ and } \psi \rightsquigarrow (f(\vec{x})\ \texttt{rdf:type}\ A) \text{ is in } \mathcal{M}\} \cup$$
$$\{(f(\vec{o})\ P\ f'(\vec{o})) \mid \vec{o} \in ans(\psi, D) \text{ and } \psi \rightsquigarrow (f(\vec{x})\ P\ f'(\vec{x})) \text{ is in } \mathcal{M}\}.$$

In this paper, we follow the materialisation approach to query answering, i.e., SPARQL queries are answered by a triplestore into which the generated KG is loaded (i.e., materialised). This is mainly motivated by the fact that the data sources we target (such as online databases with API access to JSON files) might not support arbitrary queries.

RML is a flexible, declarative language designed for specifying (GAV) mappings from heterogeneous data sources to RDF. Building on R2RML, it extends support beyond relational databases to handle JSON, XML, and CSV formats. Its modular architecture [22] includes the RML-core module[3], which inherits the core mechanisms of R2RML for RDF triple generation, and the RML-IO module[4], which enables access to diverse data sources by defining *logical sources* and providing a relational abstraction for them.

The use of LLM queries in RML mappings is possible thanks to the RML-FNML module[5], which enables the specification of user-defined functions within triple maps (i.e., in the target of GAV-style mappings). We can distinguish between two disjoint classes of function symbols: *(i) uninterpreted* function symbols (e.g., IRI templates), and *(ii)* function symbols with associated semantics (e.g., `concat` computing the concatenation of two strings, or an LLM query).

For conciseness and readability, throughout the paper, we adopt the mapping language of Ontop, a popular Virtual Knowledge Graph system [23], for concrete examples of (RML) mapping rules. This language closely aligns with the abstract syntax introduced in the previous section. Below is an example of a mapping assertion in this syntax:

```
source    SELECT m_id, name FROM medicines
target    :med-{m_id} rdf:type :Drug .
          :med-{m_id} :prefLabel {name} .
```

This example corresponds to two GAV-style mapping assertions. The template `:med-{m_id}` denotes an *uninterpreted unary function symbol* applied to the database attribute `m_id`, while `:Drug` and `:prefLabel` are IRIs in the target RDF graph. Assuming a database instance where the SQL query in the `source` clause returns the tuple $(\texttt{m_id} \mapsto \texttt{"001-100"}, \texttt{name} \mapsto \texttt{"Amoxicillin"})$, the mapping generates the following RDF triples[6]:

```
:med-001-100 rdf:type :Drug .
:med-001-100 :prefLabel "Amoxicillin" .
```

[3] https://kg-construct.github.io/rml-core/ontology/documentation/index-en.html
[4] https://kg-construct.github.io/rml-io/ontology/documentation/index-en.html
[5] https://kg-construct.github.io/rml-fnml/ontology/documentation/index-en.html
[6] Expressed in Turtle syntax (https://www.w3.org/TR/turtle/).

3. LLM-Enhanced Mappings with RML

The novel idea of this paper is to integrate LLMs into OBDA mappings so as to extract more structured information from semi-structured data. Our approach leverages the expressiveness of the RML-FNML module, which permits the specification of arbitrarily complex user-defined functions within mapping assertions. This integration enables the dynamic extraction of structured facts from unstructured fields at mapping time, while preserving the declarative nature of RML mappings. In this section we present it conceptually. Then in Section 4 we instantiate it on a medicines information use-case.

To begin with, we define a logical source S over the raw data (JSON documents in our case). For our purposes, S can be viewed as an instance of a relation over a fixed schema, where the attributes correspond to references specified in the RML mappings for S. In S, we distinguish *structured attributes*, the values of which can be directly mapped to RDF terms using standard RML templates, and *unstructured attributes*, the values of which are free-text fields in natural language, so may require natural language processing to meaningfully map them to RDF terms.

We assume a fixed LLM Λ. For simplicity, we abstract it as a total deterministic function from strings to strings, where $\Lambda(s)$ denotes the output of Λ on an input string s. Furthermore, we introduce a number of user-defined function symbols, such as `getAnswerFromLLM` and `getFloatFromLLM`, with semantics parameterised by Λ. These function names are interpreted by querying Λ and then extracting the answer of the expected type from the output of Λ. For example, assuming that `toJSONObject` converts a string to a JSON object, the output of Λ is of the form "`{"answer": ". . . "}`", and for a JSON object $o = \{k\colon v\}$, the expression $o.k$ evaluates to v, the semantics of `getAnswerFromLLM` on a string s can be defined as:

$$\texttt{getAnswerFromLLM}_\Lambda(s) := \texttt{toJSONObject}(\ \Lambda(s)\ ).\texttt{answer}$$

Now, for each property P the value of which should be extracted from an unstructured attribute U, we assume to have a prompt template π_P—an instruction string or a few-shot example. Then the expression

$$\texttt{getAnswerFromLLM}(\ \texttt{concat}(\pi_P,\ U)\ )$$

may appear as a term in the target of a mapping assertion for P. Its semantics is as one would expect, with U evaluating to the textual content of attribute U for a given tuple in S.

Finally, the definition of the generated KG $\mathcal{M}(D)$ should be extended to take into account Λ. It is straightforward to do so; we omit it due to the lack of space.

Example Suppose S contains an attribute `doseInfo` and we want to extract from it the value for property `hasDoseAmount`. This can be done in an RML triple pattern of the form:

```
:med-{m_id} :hasDoseAmount getAnswerFromLLM(concat(PROMPT, {doseInfo})) .
```

where PROMPT is the string :

```
Produce output as a JSON string of format {"answer": DOSAGE_AMOUNT}.
Extract dosage amount:
```

For a value "1 g, every 4–6 hours; maximum 4 g per day" of `doseInfo` and 001-100 of a structured attribute `m_id`, the generated RDF triple could be:

```
:med-001-100 :hasDoseAmount "1" .
```

4. Medicines Information Use-Case with LLM-Enhanced RML

In this section we demonstrate our approach on a medicines information use-case. We start with a motivation for the use-case in Section 4.1 and then in Section 4.2 describe our full OBDA setup. All our artefacts and code can be found at https://github.com/pharmaKG/rmlai.

4.1. Medicines Information

Annually, there is an estimated 66 million medication errors occurring within the NHS that are harmful to patients. Each year, avoidable adverse drug events are estimated to cause or contribute to a thousand

```
{
    "identifier": "apixaban",
    "title": "Apixaban",
    "breastFeeding": {
        "specificInfo": "A risk cannot be excluded: animal studies shows presence in milk."
    },
    ...
    "sideEffects": {
        "specificInfo": "Common: haemorrhage, contusion, epistaxis, haematoma. Uncommon: thrombocytopenia,
             hypotension, post procedural haematoma."
    },
    "therapeuticPlan": [{
        "indications": ["Treatment of deep-vein thrombosis", "Treatment of pulmonary embolism"],
        "dosages": [{
            "id": "4b59244d-c27d-4be2-a954-6a793f1c7cb0",
            "patientGroup": "adult",
            "detailedPatientGroup": "Adult",
            "doseInfo": "10 mg taken twice daily for 7 days.",
            "routeOfAdministration": "By mouth"
        }]
    }]
}
```


Figure 1: A fragment of the JSON document about Apixaban from our dataset.

deaths [24]. The use and application of medicines are often complex, requiring choosing the right drug, its dosage, administration modality and considering interaction with the existing treatments or conditions [25]. For instance, a nurse may need to know if a medication can be administered through a nasogastric tube or if a particular drug is safe to use while breastfeeding. A consultant might inquire about the occurrence of rare side effects.

To ensure proper use of medicines, medicines information (MI) departments exist to assist healthcare staff and patients with their queries [26]. When an MI department receives a query, it is their responsibility to sift through a number of dedicated online databases [10, 27, 28] to find accurate answers, which then should be compiled into a document including all relevant references. A limitation of this manual process is that even for relatively straightforward questions, it requires significant time, as each database must be reviewed individually, and the answer must be properly collated and referenced.

Automating the MI delivery service, at least for basic enquiries, has the potential to reduce the number of medication errors by providing access to timely and accurate medicines information to health professionals. We demonstrate how our approach can be used towards this automation when the medicines data comes as semi-structured information in JSON format. Our idea is to extract a KG from the data using OBDA mappings, so that answering MI enquiries would amount to answering SPARQL queries over the extracted KG.

4.2. The OBDA Setup

We describe our OBDA instantiation of the medicines information use-case.

Data. We assume that information about drugs is available in JSON format, where some part of information is structured, while some information is given in the form of plain text, similarly to the online databases such as BNF [10] and EMC [27]. For the experiments, we created JSON documents, each stored in a separate file on the local file system, about 20 drugs providing (a subset of) information on their usage (including patient group, indications, dosage, frequency), side-effects, as well as advice for patients that are breastfeeding, pregnant, or have hepatic or renal impairment.

A fragment of one of such documents is shown in Figure 1. Here, the route of administration in dosages is structured, given that it takes one of the possible values, “By mouth”. However the dosage amount and frequency are stated in one string “10 mg taken twice daily for 7 days”. Advice for special categories of patients and the side effects are unstructured as well.

We note that for the sake of our small scale experiment, there is always at most one entry in the therapeutic plan and at most one entry in dosages. This does not restrict the scope of our methodology.

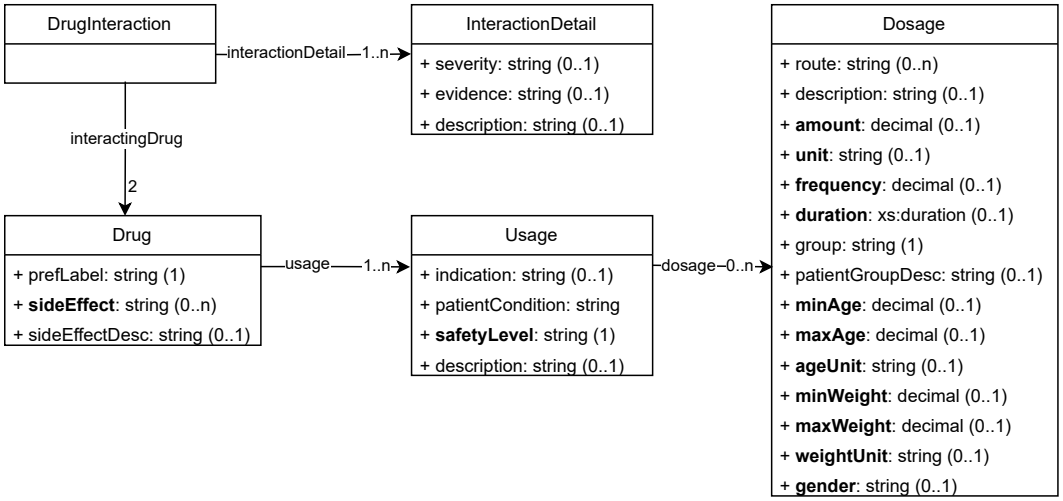


Figure 2: The UML class diagram for the Medicines Information ontology.

Also, we created auxiliary identifiers at various levels of the JSON documents to simplify the subsequent linking of different entities (see, e.g., the `id` field inside `dosages`).

Ontology. We built a simple ontology to capture the data in our dataset along with a SHACL file that defines the schema of a valid RDF graph. There are 5 classes and 24 properties, see Figure 2 for an overview. For every drug there may be several usages, one for each indication and patient condition. Usually, one usage is reserved for patients without special conditions. Further, there may be up to 4 different usage instances: for pregnant, breastfeeding and patients with hepatic or renal impairment. These typically include a safety advice. Then, each usage is associated with dosage instances, one for each patient group. Dosage information includes the route of administration, the amount, unit, frequency and duration of the dosage, as well as the information pertaining to the patient group such as gender and the age and weight ranges.

Note that the values for properties highlighted in bold are those that need to be extracted from unstructured text. We refer to those properties as *LLM-extracted* properties. All IRIs and the rest of the values can be extracted directly from JSON. We refer to properties the value for which could be mapped directly from JSON as *immediately accessible*. When an entity has LLM-extracted properties, the original unstructured string is stored in one of the `description` or `xxxDesc` properties.

Mappings We created RML mapping rules for all our classes and properties. For each class and immediately accessible property, we created standard rules. For each LLM-extracted property, the mapping rule involves a call to a user-defined function implementing an LLM query. For `sideEffect` and `safetyLevel` prompts, we adopted a zero-shot prompting strategy, while for the remaining properties a few-shot prompting strategy, where each prompt included few examples consisting of an input and its corresponding expected output. The choice of the strategy was made based on initial testing; a more thorough evaluation of different prompting strategies could be included in a future work.

Below we provide mapping rules for the `Dosage` class, the `route` and `amount` properties (instantiated for apixaban). For readability, we present them using the more concise Ontop-like notation:

```
source json_to_table("$.therapeuticPlan[*].dosages[*]", apixaban.json)

target http://drug.uk/dosage/{id} rdf:type drug:Dosage ;
                                  drug:route {routeOfAdministration};
                                  drug:amount grel:getFloatFromLLM(
                                                  grel:concat(DOSAGE_PROMPT, {doseInfo})
                                              ) .
```

The `source` declaration in the first line essentially creates a relation with the attributes `(id, patientGroup, detailedPatientGroup, doseInfo, routeOfAdministration)`, through a JSON-

path[7] query over the file "apixaban.json". Each tuple in this relation corresponds to an element of the `dosages` array (see Figure 1). In the `target` part, the first two rules are standard R2RML/RML rules. The rule for the `amount` property states that the object value is computed as `grel:getFloatFromLLM`(p), where

- p is the string obtained as the concatenation of the fixed string DOSAGE_PROMPT with the value of `doseInfo`.
- `grel:getFloatFromLLM` is a user-defined function returning a decimal number. Its definition is provided in the `functions.ttl` file that defines standard and custom functions used within RML mapping rules to transform, enrich or filter data during the mapping process. In particular, this file specifies the Java class and the name of the method within it implementing the actual call to the LLM and extraction of the floating point value from its reply.
- DOSAGE_PROMPT is the following prompt template:

```
task: Extract dosage amount from the provided text.
output: a json string of format: {"response": DOSAGE_AMOUNT}. DO NOT RETURN ADDITIONAL EXPLANATIONS OR TEXT
      OR MARKDOWN FORMATTING. Only JSON as string.
rule: DOSAGE_AMOUNT is a float value. And it should be a valid xsd:decimal. If no relevant information is
     found in the text, DOSAGE_AMOUNT is null. Do not put null within double quotes.
example: if the text says 50mg daily, DOSAGE_AMOUNT is 50. if it says 50mg in 2 divided dose, as 50 divided
      by 2 is 25, DOSAGE_AMOUNT is 25. if text says apply 1 millimeter, DOSAGE_AMOUNT is 1.
text:
```

5. Evaluation

In this section we evaluate our approach in terms of correctness of the generated knowledge graph and of the answers to SPARQL queries over this graph. For all experiments, we employed three versions of varying sizes of DeepSeek-R1-Distill-Qwen, an open-source LLM with reasoning capabilities [29], with 7, 14 and 32 billion parameters. We chose these models because they could be installed and run locally, whereas their increasing sizes allowed us to explore the trade-off between their computational cost and performance. All experiments were run on Intel Xeon Gold 5317 CPU with 32 cores and NVIDIA A100 GPU with 80GB memory.

5.1. Correctness of the Generated Knowledge Graph

We report our findings on the quality of the KGs generated using our OBDA setup, RMLMapper v7.3.3, and the three LLMs.

For each LLM, we repeated the KG generation process three times to account for possible differences in its outputs. Since for our tasks we were interested in a more deterministic behaviour of the LLMs, we set the temperature parameter to 0. Typically, lower values of the temperature parameter correspond to more deterministic behaviours of LLMs: 0 forces an LLM to output a word with the highest probability. In our experiments, this ensured that all three runs for each LLM were identical. In what follows, all statistics refer to one run for an LLM.

We now discuss the correctness of the LLM-extracted values (the ground truth was established manually). Across 20 drugs, the total number of triples in the extracted RDF graph was 14,641. Out of these, the object value of 963 triples was LLM-extracted. For each LLM, we summarise the number and ratio of errors within LLM-extracted facts, as well as the ratio of errors in the context of the complete RDF graph, and report the average generation time in Table 1. We also present a breakdown of the errors per property together with the count of values to be extracted using LLM. The smallest model performed worst (0.25% error rate for LLM-extracted values and 0.16% error rate for the whole graph), and one run took around 13 minutes. The two bigger models performed similarly (0.05% error rate for LLM-extracted values and 0.003% error rate for the whole graph), while DeepSeek-14b was about twice faster than DeepSeek-32b.

[7] https://en.wikipedia.org/wiki/JSONPath

	DeepSeek-7b	DeepSeek-14b	DeepSeek-32b
Number and ratios of errors	237 / 0.25% / 0.16%	50 / 0.05% / 0.003%	53 / 0.05% / 0.003%
Generation time	13min	20min	40min
safetyLevel / 80	22	3	9
amount / 19	2	1	1
unit / 19	2	2	1
frequency / 18	2	1	1
duration / 19	6	2	2
sideEffect / 790	203	41	39

Table 1
Error number, ratio of errors out of extracted values and out of all RDF triples in the generated RDF graph, and the generation time, per LLM (1 run). We also show the breakdown of errors per property; the number of values to be extracted is shown next to the property name.

We cautiously consider our approach to be successful at generating "almost correct" knowledge graphs from semi-structured information: 50 errors in 14,641 facts amount to 0.003% error rate. Although we do not include a comparison with a pure LLM-based solution (left for future work), we conjecture that such a solution would either produce significantly more errors than ours, or would require relying on more advanced, and hence more expensive, LLMs. Another advantage of our solution is that it guarantees the conformance of the generated graph with the ontology. For each LLM, the generated RDF graph has successfully passed SHACL validation for the portion of the data available in our JSON dataset. Whereas a pure LLM-based solution may produce RDF triples formulated using wrong vocabulary or not satisfying integrity constraints.

5.2. Answers to queries

We now evaluate our approach in terms of SPARQL query answers over the generated RDF graphs, and compare it to three other solutions to accessing data: *(i)* SPARQL queries evaluated over the KG generated using 'standard' (i.e., not LLM-enhanced) RML mappings, referred to as SPARQL-over-RML-KG; *(ii)* Natural language queries answered by LLM, referred to as NL-over-LLM; *(iii)* Natural language queries answered by LLM, each over a single JSON document, referred to as NL-over-JSON. In NL-over-LLM, we are essentially probing the knowledge acquired by LLM from the training data, while NL-over-JSON follows a RAG-like (retrieval augmented generation) architecture, where LLM is expected to pull relevant information from a provided document, a JSON document about a drug in our case.

We selected 28 queries, loosely based on actual MI enquiries, to be answerable over our dataset. Each query is over a specific drug. The first 8 queries are expected to return one of three values *Yes*, *No* or *Unknown*. The latter value should be returned when there is no evidence to back up a positive or a negative answer. The remaining 20 queries ask the dosage amount and unit for an indication, for each of the 20 drugs in our dataset. The correct answers were determined by evaluating the queries of the ground truth graph.

We employed zero-shot prompting strategy for the LLM queries in NL-over-LLM and NL-over-JSON. Based on our initial experiments, we asked LLMs to provide an answer in a structured format. Below are the zero-shot instructions we used for the dosage queries:

```
Rule:
- Provide the answer in a structured plain text JSON format
- JSON should contain "dosage amount" and "dosage unit" as keys and their respective values.
- Do not include any additional information or explanation.
- Do not provide JSON in markdown formatting. Only provide plain text JSON.
```

For SPARQL-over-RML-KG, there were two correct *Yes* answers, all other answers were *Unknown* since the relevant data was not accessible when using standard RML mappings. Six of the unknown answers coincided with the ground truth, therefore we consider that SPARQL-over-RML-KG returned

	DeepSeek-7b	DeepSeek-14b	DeepSeek-32b
NL-over-LLM	24 / 110s	21 / 180s	21 / 300s
NL-over-JSON	14 / 90s	8 / 150s	6 / 330s
SPARQL-over-RML-LLM-KG	3	2	1

Table 2
Number of incorrect query answers out of 28, per query answering approach and LLM. For the natural language queries, we report the average time it took to answer all 28 queries with LLMs.

20 incorrect answers. As for the approaches involving calls to LLMs, either for KG construction or for direct query answering, we summarise the numbers of incorrect query answers in Table 2. Asking for answers directly from LLMs results in over 20 errors for all three models. Providing LLMs relevant information in a JSON file reduces the number of errors to 14 for the smallest, 8 for the medium and 6 for the largest models. The number of errors in SPARQL query answers over the three generated graphs ranges from 3 for the smallest LLM, to 2 for the medium and 1 for the largest one.

This simple comparison shows that our approach advances the existing work along two dimensions. First, with respect to the standard RML mappings, our LLM-enhanced mappings make more data accessible for structured query answering, whose benefits include better explainability of the answers. Second, providing LLMs with fine-grained context in LLM-enhanced mappings can reduce the risk of incorrect answers, comparing to probing the implicit knowledge of an LLM as in NL-over-LLM, or to providing a potentially much bigger context as in NL-over-JSON. Finally, though not currently implemented, our approach allows for easy verification of query answers, by providing to the user the original (relatively short) unstructured text from which the values involved in deciding the query outcome were extracted. Arguably, verification in this case is more straightforward than in NL-over-LLM or NL-over-JSON, where the user would have to look at a whole (possibly quite large) JSON document.

6. Conclusions

In this paper, we presented a promising approach for generating Knowledge Graphs (KGs) from semi-structured data by integrating LLM queries within RML mappings. Our method improves correctness of query answers as opposed to an LLM-based solution to query answering, as demonstrated through our preliminary experiments.

In the future, we plan to address the limitation of RMLMapper that prevents the simultaneous extraction of semantically related property values, requiring each property to be extracted individually. This may increase the risk of inconsistencies when these values are interpreted collectively. For instance, "1 dose every 6 hours" is equivalent to "4 doses every day". Therefore, two different LLM queries might extract 1 as frequency and 1 as duration, resulting in an incorrect pair. Moreover, this constraint increases the number of LLM queries, which in turn increases the overall KG generation time. To address these issues, future work will explore using LLMs in the source component of mapping assertions, e.g., by relying on the RML-LV module[8], which introduces the notion of *logical views* specifying virtual relational schemas over data, and consequently allows for a two-stage mapping process.

In our experiments we materialised the whole KG. This would not be feasible in real life when offering a fully-fledged solution to accessing online databases. Therefore, future work could look into materialising only the portion of the KG relevant to answering the current query.

Acknowledgements This research was partially supported by the HEU project CyclOps (grant agreement n. 101135513), by the Province of Bolzano and FWF through the project Ontegra (DOI 10.55776/PIN8884924), by the Province of Bolzano and EU through the project EFRE/FESR 1078 CRIMA, and by the Italian PRIN project S-PIKACHU.

[8] https://kg-construct.github.io/rml-lv/ontology/documentation/index-en.html

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. on Data Semantics (2008) 133–173. doi:10.1007/978-3-540-77688-8_5.

[2] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, W3C, 2014. Available at http://www.w3.org/TR/rdf11-concepts/.

[3] S. Harris, A. Seaborne, SPARQL 1.1 Query Language, W3C Recommendation, W3C, 2013. Available at http://www.w3.org/TR/sparql11-query.

[4] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation, W3C, 2012. Available at http://www.w3.org/TR/r2rml/.

[5] E. Botoeva, D. Calvanese, B. Cogrel, J. Corman, G. Xiao, Ontology-based data access - beyond relational sources, Intelligenza Artificiale 13 (2019) 21–36.

[6] D. Chaves-Fraga, E. Ruckhaus, F. Priyatna, M.-E. Vidal, O. Corcho, Enhancing virtual ontology based access over tabular data with morph-csv, Semantic Web 12 (2021) 869–902. doi:10.3233/SW-210432.

[7] A. Dimou, M. V. Sande, J. Slepicka, P. A. Szekely, E. Mannens, C. A. Knoblock, R. V. de Walle, Mapping hierarchical sources into RDF using the RML mapping language, in: Proc. of ICSC, IEEE, 2014, pp. 151–158. URL: https://doi.org/10.1109/ICSC.2014.25. doi:10.1109/ICSC.2014.25.

[8] A. Iglesias-Molina, D. V. Assche, J. Arenas-Guerrero, B. D. Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, A. Dimou, Rml ontology portal: Modular resources for the rdf mapping language, https://kg-construct.github.io/rml-resources/portal/, 2023. Accessed: 2025-06-05.

[9] RML.io, RMLMapper: A java implementation for executing RML rules to generate Linked Data, https://github.com/RMLio/rmlmapper-java, 2025. Version 7.3.3, MIT License.

[10] Joint Formulary Committee, British national formulary, https://bnf.nice.org.uk/, Accessed: 2025-05-15.

[11] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, D. Roth, Recent advances in natural language processing via large pre-trained language models: A survey, ACM Computing Surveys 56 (2023) 1–40.

[12] K. Hakala, S. Pyysalo, Biomedical named entity recognition with multilingual BERT, in: Proc. of the 5th Workshop on BioNLP Open Shared Tasks, Association for Computational Linguistics, 2019, pp. 56–61. URL: https://aclanthology.org/D19-5709/. doi:10.18653/v1/D19-5709.

[13] C. Alt, M. Hübner, L. Hennig, Improving relation extraction by pre-trained language representations, arXiv preprint arXiv:1906.03088 (2019).

[14] L. Yao, C. Mao, Y. Luo, Kg-bert: Bert for knowledge graph completion, arXiv preprint arXiv:1909.03193 (2019).

[15] I. Melnyk, P. Dognin, P. Das, Grapher: Multi-stage knowledge graph construction using pretrained language models, in: NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications, 2021.

[16] T. Ayoola, S. Tyagi, J. Fisher, C. Christodoulopoulos, A. Pierleoni, ReFinED: An efficient zero-shot-capable approach to end-to-end entity linking, in: A. Loukina, R. Gangadharaiah, B. Min (Eds.), Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track, Association for Computational Linguistics, Hybrid: Seattle, Washington + Online, 2022, pp. 209–220. URL: https://aclanthology.org/2022.naacl-industry.24/. doi:10.18653/v1/2022.naacl-industry.24.

[17] G. Ciatto, A. Agiollo, M. Magnini, A. Omicini, Large language models as oracles for instantiating ontologies with domain-specific knowledge, Knowledge-Based Systems 310 (2025) 112940. URL:

https://www.sciencedirect.com/science/article/pii/S0950705124015740. doi:https://doi.org/10.1016/j.knosys.2024.112940.

[18] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, et al., A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, ACM Transactions on Information Systems 43 (2025) 1–55.

[19] G. Xiao, L. Ren, G. Qi, H. Xue, M. D. Panfilo, D. Lanti, LLM4VKG: Leveraging large language models for virtual knowledge graph construction, in: Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI), 2025. To appear.

[20] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, S. Flesca, The lixto data extraction project: back and forth between theory and practice, in: Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '04, Association for Computing Machinery, New York, NY, USA, 2004, p. 1–12. doi:10.1145/1055558.1055560.

[21] D. Lembo, F. M. Scafoglieri, Ontology-based document spanning systems for information extraction, Int. J. Semantic Comput. 14 (2020) 3–26. doi:10.1142/S1793351X20400012.

[22] A. Iglesias-Molina, D. V. Assche, J. Arenas-Guerrero, B. D. Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, A. Dimou, The RML ontology: A community-driven modular redesign after a decade of experience in mapping heterogeneous data to RDF, in: Proc. ISWC, volume 14266 of *LNCS*, Springer, 2023, pp. 152–175. URL: https://doi.org/10.1007/978-3-031-47243-5_9. doi:10.1007/978-3-031-47243-5_9.

[23] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering SPARQL queries over relational databases, Semantic Web J. (2016). DOI: 10.3233/SW-160217.

[24] R. A. Elliott, E. Camacho, D. Jankovic, M. J. Sculpher, R. Faria, Economic analysis of the prevalence and clinical and economic burden of medication error in england, BMJ Quality & Safety 30 (2021) 96–105. URL: https://qualitysafety.bmj.com/content/30/2/96. doi:10.1136/bmjqs-2019-010206.

[25] G. P. Velo, P. Minuz, Medication errors: prescribing faults and prescription errors, British journal of clinical pharmacology 67 (2009) 624–628.

[26] J. Rutter, R. Fitzpatrick, P. Rutter, What effect does medicine advice provided by uk m edicines i nformation pharmacists have on prescriber practice and patient care: a qualitative primary care study, Journal of evaluation in clinical practice 21 (2015) 307–312.

[27] Electronic medicines compendium, https://www.medicines.org.uk/emc/, Accessed: 2025-05-15.

[28] Medusa, the nhs injectable medicines guide, https://www.medusaimg.nhs.uk/, Accessed: 2025-05-15.

[29] DeepSeek-AI, Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL: https://arxiv.org/abs/2501.12948. arXiv:2501.12948.