RML Meets LLMs: More Structure, Fewer Errors

Author Name
Affiliation
email@example.com

Abstract

Declarative mapping languages such as RML generate knowledge graphs from structured data, but much real-world data is semi-structured, combining structured fields with unstructured natural language. We propose integrating LLM queries directly into RML mapping rules: structured fields are handled by standard RML, while LLMs extract ontological terms from unstructured text. Feeding LLMs only short, focused snippets reduces errors compared to end-to-end LLM solutions, while preserving correctness guarantees for the structured part. We instantiate and evaluate our approach on a Medicines Information use-case with pharmaceutical JSON data, assessing the correctness and completeness of the resulting knowledge graphs and query answers against pure LLM-based and pure RML-based solutions.

1 Introduction

The Ontology-Based Data Access (OBDA) framework [Poggi *et al.*, 2008] 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) [Cyganiak *et al.*, 2014]. 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 [Harris and Seaborne, 2013]. 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 [Das *et al.*, 2012]. With the advent of non-relational databases, the OBDA framework has been extended to accommodate alternative data models [Botoeva *et al.*, 2019; Chaves-Fraga *et al.*, 2021]. 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 [Dimou *et al.*, 2014; Iglesias-Molina *et al.*, 2023b], the RDF Mapping Language. RMLMapper is a library for generating knowledge graphs from various data formats, including CSV, JSON, and XML [RML.io, 2025].

A fundamental assumption when generating knowledge graphs following the OBDA principle is that data are inherently structured. However, a significant portion of real-world data is, at best, semi-structured. As a running example, consider online databases that catalogue medicines [Joint Formulary Committee, Accessed 2025 05 15]. Notably, information such as the name of a drug or its interactions with other drugs is structured, but other information may come as unstructured text. For instance, "1 g, every 4–6 hours; maximum 4 g per day" is a dosage information for paracetamol; such instructions may come in a variety of formats, even for the same drug, e.g., "1 g, every 4–6 hours, dose 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.

The standard solution for unstructured text is to employ large language models (LLMs)–natural language processing tools that excel at understanding and generating language [Min *et al.*, 2023]. 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 [Hakala and Pyysalo, 2019; Alt *et al.*, 2019; Yao *et al.*, 2019; Melnyk *et al.*, 2021; Ayoola *et al.*, 2022; Ciatto *et al.*, 2025]. Despite these advances, however, the main challenge remains factual correctness of the extracted assertions, due to the so-called *hallucinations* [Huang *et al.*, 2025]–a consequence of LLMs being statistical models that

output words with high probability of continuing the current phrase, rather than verified facts. A pure LLM-based solution may therefore not be acceptable in high-stakes applications, where the knowledge graph is expected to be factually correct, or query answers 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 integrating LLM queries within RML mappings, where the input to the LLM is computed by the source part of a mapping rule and its output is used to generate RDF terms. Our approach has several advantages 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 handled by standard RML rules. Second, the LLM is fed only short, focused text snippets, reducing the risk of confusing it. Finally, verification of the correctness of the knowledge graph and query answers is more straightforward, as each LLM generated fact can 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 used LLMs to generate mappings automatically [Xiao *et al.*, 2025]. Other information extraction techniques have been used to convert textual documents to structured representations [Gottlob *et al.*, 2004; Lembo and Scafoglieri, 2020], 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 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 [Harris and Seaborne, 2013].

[1]An unnamed resource

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

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* (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 [Iglesias-Molina *et al.*, 2023a] 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)* func-

[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

tion 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 [Calvanese *et al.*, 2016], 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 `(m_id` $\mapsto$ `"001-100",` `name` $\mapsto$ `"Amoxicillin")`, the mapping generates the following RDF triples[6]:

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

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 and in Section 4 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.

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 deaths [Elliott *et al.*, 2021]. 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 [Velo and Minuz, 2009]. 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 [Rutter *et al.*, 2015]. When an MI department receives a query, it is their responsibility to sift through a

[6]Expressed in Turtle syntax (`https://www.w3.org/TR/turtle/`).

```
{
    "identifier": "apixaban",
    "title": "Apixaban",
    "breastFeeding": {
        "generalInfo": "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 orally twice daily for 7 days.",
            "routeOfAdministration": "By mouth"
        }]
    }]
}
```


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

number of dedicated online databases [Joint Formulary Committee, Accessed 2025 05 15; EMC, Accessed 2025 05 15; Med, Accessed 2025 05 15] 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 [Joint Formulary Committee, Accessed 2025 05 15] and EMC [EMC, Accessed 2025 05 15]. 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 orally 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. 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 corre-

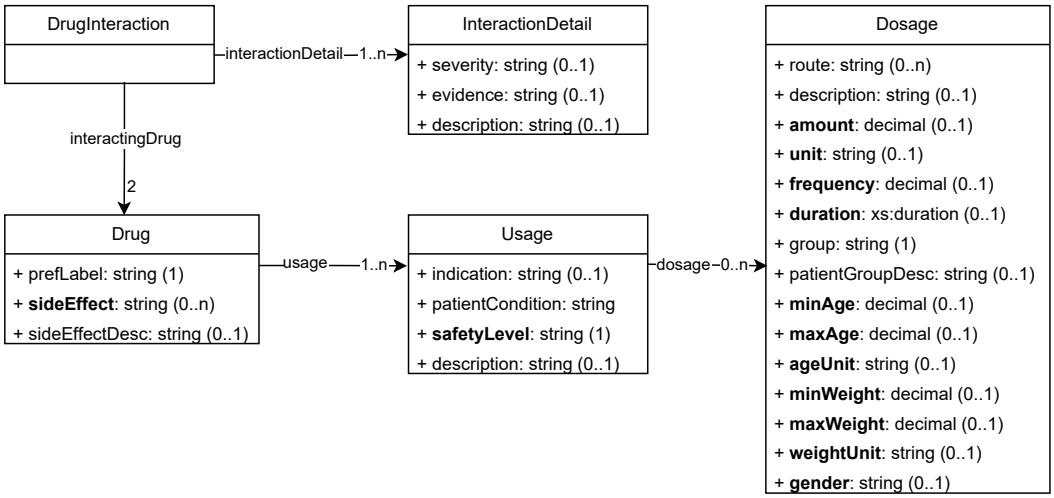

Figure 2: The UML class diagram for the Medicines Information ontology.

sponding 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` and `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, defined in `functions.ttl`, a file specifying standard and custom functions used in RML mappings. It specifies the Java class and the method that implements the LLM call and extracts 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:
```

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

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 our core experiments, we employed three versions of varying sizes of DeepSeek-R1-Distill-Qwen, an open-source LLM with reasoning capabilities [DeepSeek-AI, 2025], 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 DeepSeek LLMs. We use two baselines: the ground truth graph (established manually) and the output of a pure LLM-based solution, in which Claude Sonnet 4.6 directly converts each JSON file into an RDF graph.

For each DeepSeek model, 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 temper-

	DeepSeek-7b	DeepSeek-14b	DeepSeek-32b	Claude Sonnet 4.6
Number and ratios of errors	79 / 0.14 / 0.05	36 / 0.06 / 0.02	34 / 0.06 / 0.02	561 / - / 0.40
Generation time	13min	20min	40min	20min
sideEffect / 372	37	18	17	392
safetyLevel / 82	18	8	6	82
amount / 20	4	1	2	6
unit / 20	4	4	3	6
frequency / 19	4	4	4	6
duration / 19	3	1	2	4
gender / 2	4	-	-	2
ageUnit / 4	1	-	-	4
weightUnit / 2	4	-	-	2
maxAge / 4	-	-	-	4
minAge / 4	-	-	-	4
minWeight / 2	-	-	-	2
usage / 100	-	-	-	11
patientCondition / 100	-	-	-	18
rdf:type / 204	-	-	-	18

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.

ature 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. Across 20 drugs, the total number of triples in the extracted RDF graph was 1,384. Out of these, the object value of 550 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, the DeepSeek columns. 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 (0.14 error rate for LLM-extracted values and 0.05 error rate for the whole graph), and one run took around 13 minutes. The two bigger models performed similarly (0.06 error rate for LLM-extracted values and 0.02 error rate for the whole graph), while DeepSeek-14b was about twice faster than DeepSeek-32b.

We now turn to the comparison with the KG generated by an LLM directly from JSON. For this task we use a more advanced model, Claude Sonnet 4.6, as the considered DeepSeek models are not large enough for a meaningful evaluation. We asked Claude to convert the JSON data into an RDF graph using the vocabulary from our ontology, letting it process each JSON file independently. This avoids combining all JSON files into one big context, potentially reducing the number of errors. We aligned the URIs generated by Claude with the URIs in the ground truth and then computed the number of different values, for each property. Those are reported in the last column of Table 1. We omit the errors in the description properties, as those do not contain structured information. The total number of errors in the Claude-generated KG is 561. It is worth noting that errors are heavily concentrated in the *sideEffect* property: 372 missing values and 20 wrong ones, as Claude does not decompose side effects into distinct triples. The *safetyLevel* property errors are due to the LLM not extracting the safety levels. While further prompting could help to reduce the error count, the pure LLM solution would nevertheless lag behind our approach on other properties.

We consider our approach successful at generating "almost correct" knowledge graphs from semi-structured information: 34 errors across 1,384 facts yield an error rate of just 2.5%. By contrast, a pure LLM-based solution–using a comparably capable model–produced substantially more errors even on small JSON files, a gap that would likely widen on more realistic, larger inputs. Another advantage of our solution is that it guarantees the conformance of the generated graph with the ontology. A pure LLM-based solution offers no such guarantee: it may produce triples with incorrect vocabulary.

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 en-

	DeepSeek-7b	DeepSeek-14b	DeepSeek-32b
NL-over-LLM	24 / 110s	22 / 180s	19 / 300s
NL-over-JSON	17 / 90s	8 / 150s	6 / 330s
SPARQL-over-RML-LLM-KG	3	2	2

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.

quiries, 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 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 19 errors for all three models. Providing LLMs relevant information in a JSON file reduces the number of errors to 17 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 the largest ones.

Our approach advances existing work along two dimensions. First, compared to standard RML mappings, LLM-enhanced mappings make more data accessible for structured query answering, with better answer explainability. Second, providing fine-grained context reduces the risk of incorrect answers compared to probing implicit LLM knowledge (NL-over-LLM) or supplying a much larger context (NL-over-JSON). Additionally, though not currently implemented, our approach allows easy verification of query answers: users can inspect the short text snippet from which relevant values were extracted, which is more straightforward than reviewing a whole JSON document as in the other approaches.

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 produces KGs of significantly better quality than a pure LLM-based solution to KG construction, and improves correctness of query answers as opposed to an LLM-based solution to query answering, as demonstrated through our preliminary experiments. Future work will focus on addressing the limitations of the current evaluation, which was done on a dataset of 20 lightweight JSON files.

In future work, we plan to address the RMLMapper limitation of extracting property values individually, which risks inconsistencies among semantically related values (e.g., "1 dose every 6 hours" vs. "4 doses every day") and increases the number of LLM queries. We will explore using LLMs in the source component of mapping assertions, e.g., via the RML-LV module[8], which introduces logical views over data enabling 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.

References

[Alt *et al.*, 2019] C. Alt, M. Hübner, and L. Hennig. Improving relation extraction by pre-trained language representations. *arXiv preprint arXiv:1906.03088*, 2019.

[Ayoola *et al.*, 2022] T. Ayoola, S. Tyagi, J. Fisher, C. Christodoulopoulos, and A. Pierleoni. ReFinED: An efficient zero-shot-capable approach to end-to-end entity linking. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 209–220. Association for Computational Linguistics, July 2022.

[Botoeva *et al.*, 2019] E. Botoeva, D. Calvanese, B. Cogrel, J. Corman, and G. Xiao. Ontology-based data access - beyond relational sources. *Intelligenza Artificiale*, 13(1):21–36, 2019.

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

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

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

[Ciatto *et al.*, 2025] G. Ciatto, A. Agiollo, M. Magnini, and A. Omicini. Large language models as oracles for instantiating ontologies with domain-specific knowledge. *Knowledge-Based Systems*, 310:112940, 2025.

[Cyganiak *et al.*, 2014] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax. W3C Recommendation, W3C, February 2014. Available at `http://www.w3.org/TR/rdf11-concepts/`.

[Das *et al.*, 2012] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language. W3C Recommendation, W3C, September 2012. Available at `http://www.w3.org/TR/r2rml/`.

[DeepSeek-AI, 2025] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[Dimou *et al.*, 2014] A. Dimou, M. Vander Sande, J. Slepicka, P. Szekely, E. Mannens, C. Knoblock, and R. Van de Walle. Mapping hierarchical sources into RDF using the RML mapping language. In *Proc. of ICSC*, pages 151–158. IEEE, 2014.

[Elliott *et al.*, 2021] R. Elliott, E. Camacho, D. Jankovic, M. Sculpher, and R. Faria. Economic analysis of the prevalence and clinical and economic burden of medication error in england. *BMJ Quality & Safety*, 30(2):96–105, 2021.

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

[Gottlob *et al.*, 2004] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The lixto data extraction project: back and forth between theory and practice. In *Proc. of PODS*, page 1–12, 2004.

[Hakala and Pyysalo, 2019] Kai Hakala and Sampo Pyysalo. Biomedical named entity recognition with multilingual BERT. In *Proc. of the 5th Workshop on BioNLP Open Shared Tasks*, pages 56–61. Association for Computational Linguistics, November 2019.

[Harris and Seaborne, 2013] Steve Harris and Andy Seaborne. SPARQL 1.1 query language. W3C Recommendation, W3C, March 2013. Available at `http://www.w3.org/TR/sparql11-query`.

[Huang *et al.*, 2025] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.

[Iglesias-Molina *et al.*, 2023a] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, and A. Dimou. The RML ontology: A community-driven modular redesign after a decade of experience in mapping heterogeneous data to RDF. In *Proc. of ISWC*, volume 14266 of *LNCS*, pages 152–175. Springer, 2023.

[Iglesias-Molina *et al.*, 2023b] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, and 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.

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

[Lembo and Scafoglieri, 2020] Domenico Lembo and Federico Maria Scafoglieri. Ontology-based document spanning systems for information extraction. *Int. J. Semantic Comput.*, 14(1):3–26, 2020.

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

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

[Min *et al.*, 2023] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pretrained language models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023.

[Poggi *et al.*, 2008] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. on Data Semantics*, pages 133–173, 2008.

[RML.io, 2025] 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.

[Rutter *et al.*, 2015] J. Rutter, R. Fitzpatrick, and P. Rutter. What effect does medicine advice provided by uk medicines information pharmacists have on prescriber practice and patient care: a qualitative primary care study. *J. of evaluation in clinical practice*, 21(2):307–312, 2015.

[Velo and Minuz, 2009] G. P Velo and P. Minuz. Medication errors: prescribing faults and prescription errors. *British J. of clinical pharmacology*, 67(6):624–628, 2009.

[Xiao *et al.*, 2025] G. Xiao, L. Ren, G. Qi, H. Xue, M. Di Panfilo, and D. Lanti. LLM4VKG: Leveraging large language models for virtual knowledge graph construction. In *Proc. of IJCAI*, 2025.

[Yao *et al.*, 2019] Liang Yao, Chengsheng Mao, and Yuan Luo. Kg-bert: Bert for knowledge graph completion. *arXiv preprint arXiv:1909.03193*, 2019.