Better Later Than Sooner: Neuro-Symbolic Knowledge Graph Construction via Ontology-grounded Post-extraction Correction

Lorenzo Loconte[1,†] l.loconte@sms.ed.ac.uk
Timothy Hospedales[1,2] t.hospedales@samsung.com
Cristina Cornelio[2] c.cornelio@samsung.com
[1]University of Edinburgh, UK [2]Samsung AI, Cambridge, UK

Abstract

Question answering (QA) is a core challenge in AI, particularly for complex queries requiring multi-hop reasoning across documents, or symbolic operations like aggregation or exhaustive listing. Retrieval-augmented generation has become the dominant approach to QA, with recent graph-based variants addressing part of these issues by organizing knowledge to better support compositional questions. However, most *textual* graph-based RAG methods still lack the structure needed for symbolic operations useful to answer complex questions reliably. This motivates *symbolic* graph-based approaches, which extract knowledge graphs (KGs) whose relations are logic predicates that enable SQL-like querying. Yet these pipelines typically use LLMs for KG extraction, which can introduce consistency issues, where extracted facts may violate commonsense ontology constraints. We propose a neuro-symbolic framework for ontology-grounded KG construction combining open-domain extraction, embedding-based canonicalization of types and predicates, and targeted LLM-based correction of ontology violations. By deferring corrections to a post-extraction stage, our method avoids repeated LLM calls, substantially reducing token usage while improving KG consistency and preserving downstream QA quality. Finally, we show that the extracted KGs are well suited for symbolic querying by measuring the occurrence of SPARQL graph patterns.

1 Introduction

An important challenge in AI is answering user questions based on new or domain-specific information, often provided as collections of text documents (Voorhees and Tice, 2000; Rajpurkar et al., 2016; Lewis et al., 2020). The most popular method for question answering (QA) is retrieval-augmented generation (RAG), which decomposes the task into two steps: retrieving text fragments relevant for a given question and using the retrieved information to generate an answer (Chen et al., 2017; Lewis et al., 2020). However, text-only QA methods often struggle with complex questions requiring multi-hop reasoning on several entities (Yang et al., 2018; Trivedi et al., 2022), symbolic operations such as arithmetic or aggregation (e.g., returning the answer that maximizes a quantity) (Dua et al., 2019; Chen et al., 2021; Zhu et al., 2021), or the composition of multiple intermediate answers (Ho et al., 2020). The challenge of answering these questions is further exacerbated by the fact that

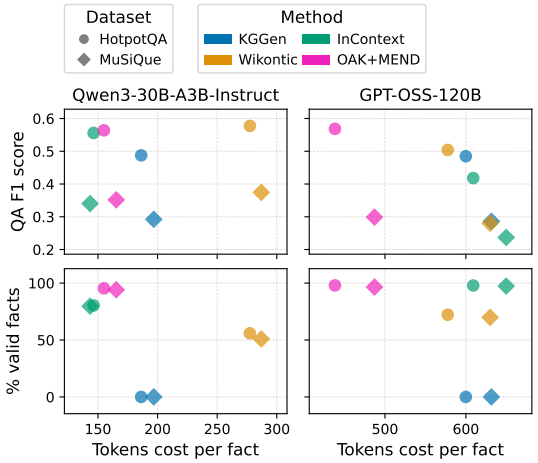


Figure 1: **Our OAK+MEND method for ontology-based KG extraction achieves better trade-offs** regarding capturing the *text semantics* (measured as QA performances, above row) and *overall ontology consistency* (measured as % of triples and qualifiers satisfying the ontology, below row), while being more efficient in terms of tokens per extracted fact (either triple or qualifier). We compare against the ontology-free KGGen (Mo et al., 2025), Wikontic (Chepurova et al., 2026) that iteratively aligns each triple to the ontology, and a baseline based on inserting the ontology as part of the LLM context during KG extraction (In-Context).

[†]The work was done during an internship at the Samsung AI Center, Cambridge (UK). Correspondence to c.cornelio@samsung.com.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 39857–39878
October 24-29, 2026 ©2026 Association for Computational Linguistics

retrieving the relevant texts becomes increasingly hard as we scale the size of the corpora (Xiong et al., 2021; Weller et al., 2026).

To address these problems, recent works have proposed structured indexes, typically graphs, which better capture global connections between documents and entities supporting multi-hop and compositional reasoning (Sarthi et al., 2024; Gutierrez et al., 2024; Gutiérrez et al., 2025; Edge et al., 2025) and also enable agentic approaches balancing exploration and exploitation on the graph index directly (Gao and Metaxas, 2026; Du et al., 2026). Among these approaches, knowledge graphs (KG) are the most prominent, encoding direct relationships between entities as subject-predicate-object triples (Peng et al., 2023; Pan et al., 2023). KGs offer a formal symbolic relational structure enabling efficient traversal as required by multi-hop questions (Ren et al., 2020; Arakelyan et al., 2021; Galkin et al., 2024; Gregucci et al., 2025).

KGs often come with an *ontology*, which specifies a domain-specific schema, including predicates, entity types, and constraints on which type of entities each predicate can link. This ontology enables verifying the consistency of the encoded information, and can guide the correction of constraint violations (Ahmetaj et al., 2022; Ferranti et al., 2024; Lin et al., 2025). Most importantly, it also makes KGs amenable to efficient retrieval through SQL-like languages such as SPARQL (Harris et al., 2013), which supports arithmetic and aggregation operations useful to answer challenging questions (Gashkov et al., 2025; Perevalov and Both, 2025).

Recent works have shown that LLMs can effectively extract KGs from text (Zhu et al., 2023; Mo et al., 2025). However, extracting KGs that comply with an existing ontology remains challenging, as it requires balancing expressivity for QA, constraints satisfaction, and tokens usage. One line of work addresses this by including ontology constraints in the LLM context during extraction (Mihindukulasooriya et al., 2023; van Cauter and Yakovets, 2024; Nie et al., 2024; Wang and Iwaihara, 2025). However, for large ontologies such as Wikidata (Vrandečić and Krötzsch, 2014), this introduces a substantial computational overhead due to the required long context size. Moreover, the consistency of the extracted KG ultimately depends on the ability of the chosen LLM to enforce the provided constraints, which makes smaller open LLMs perform worse. Although we can leverage ad-hoc models to retrieve relevant ontology fragments for each text, these must be trained for the specific ontology considered (Zhang and Soh, 2024). Another line of work employs LLMs to extract open-domain triples, which are then iteratively aligned to the ontology using multiple LLM calls (Arun et al., 2025; Lu et al., 2025; Chepurova et al., 2026). While this offers flexibility in encoding relevant information, it requires many LLM calls per extracted triple, leading to a significant cost in terms of tokens.

Contributions. **(i)** We propose an efficient method for KG construction from text under ontology constraints that does not insert the constraints into the LLM context, but rather exploits text embeddings to map the extracted entity types and predicates to the ontology. **(ii)** We symbolically detect the triples and qualifiers that violate the ontology and selectively correct each of them by prompting an LLM with a candidate set of corrective actions. This allows us to efficiently correct one or many triples (or qualifiers) with a single LLM call. **(iii)** Our experiments show our method to be more token efficient than existing baselines, yet achieving ontology consistency rates for triples and qualifiers of up to 98.4% and 96.8% respectively. QA experiments further show that our post-extraction corrections preserves crucial information from the text. Then, to assess how amenable the extracted KGs are to symbolic querying via SPARQL, **(iv)** we introduce a benchmark based on graph patterns.

Code and data are available at `https://github.com/corneliocristina/OAK_MEND`.

2 Background: Ontology Constraints

A KG $\mathcal{G}$ is a directed graph where links are encoded using subject-predicate-object triples: $\mathcal{G} = \{(s_i, r_i, o_i)\}_i \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ where $\mathcal{E}$ and $\mathcal{R}$ denote the sets of entities and predicates, respectively. KGs may also include qualifiers useful to refine or contextualize the semantics of triples. Given a set of qualifier predicates $\mathcal{Q} \subset \mathcal{R}$, a qualifier is a pair (r_q, o_q), with $r_q \in \mathcal{Q}$, $o_q \in \mathcal{E}$, that is associated to a single triple. For instance, the sentence "Alan Turing earned his Ph.D. at Princeton University in 1938" can be modeled as the triple $(\text{Alan Turing}, \text{educated at}, \text{Princeton Univ.})$ equipped with the qualifier $(\text{point in time}, 1938)$. A key component of a KG is its *ontology*, which define the admissible entity types and predicates, as well as the constraints they must satisfy. Following the Wikidata data model (Vrandečić and Krötzsch, 2014), we consider an ontology consisting of (1)

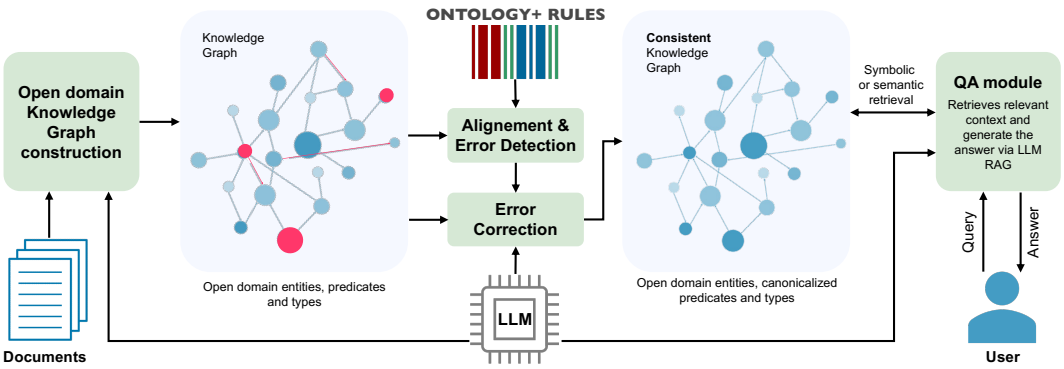


Figure 2: **Overview of the OAK+MEND method.** Documents are processed by an LLM to construct an open-domain knowledge graph, then aligned to the ontology by canonicalizing entity types and predicates. Symbolic rules derived from the ontology detect domain-range violations (both for triples and qualifiers). Detected inconsistencies are corrected through LLM calls, producing a consistent KG for (symbolic/semantic) retrieval for, e.g., QA tasks.

a hierarchy of entity types, (2) domain-range constraints, and (3) constraints defined over qualifiers. Below we formalize each ontology component.

Entity types hierarchy. Let $\mathcal{T}$ be a set of entity types, and let $t_1 \prec t_2$ denote the "is-subclass-of" partial ordering between types $t_1, t_2 \in \mathcal{T}$. Moreover, for each entity $e \in \mathcal{E}$ we denote by $\tau(e) \subset \mathcal{T}$ the set of its associated types. For example, $\mathsf{mathematician} \in \tau(\mathsf{Alan\ Turing})$ states that Alan Turing is typed as mathematician, while $\mathsf{mathematician} \prec \mathsf{human}$ specifies that mathematician is a subclass of the type human.

Domain-range constraints. For each predicate $r \in \mathcal{R}$, let $\mathsf{dom}(r) \subseteq \mathcal{T}$ and $\mathsf{rng}(r) \subseteq \mathcal{T}$ denote the *domain* and *range* constraints of r, respectively. These specify the admissible subject and object types for triples involving r. A triple (s, r, o) satisfies the domain constraint if $t_s \prec u_s$ for some $t_s \in \tau(s), u_s \in \mathsf{dom}(r)$, and satisfies the range constraint if $t_o \prec u_o$ for some $t_o \in \tau(o), u_o \in \mathsf{rng}(r)$. For example, given

$$\mathsf{dom}(\mathsf{educated\ at}) = \{\mathsf{human}\}$$
$$\mathsf{rng}(\mathsf{educated\ at}) = \{\mathsf{university}\}, \text{then}$$

$(\mathsf{Alan\ Turing}, \mathsf{educated\ at}, \mathsf{Princeton\ University})$ satisfies the domain constraint of educated at since $\mathsf{mathematician} \prec \mathsf{human}$; and the range constraint whenever $\mathsf{university} \in \tau(\mathsf{Princeton\ University})$.

While prior work Feng et al. (2024); Chepurova et al. (2026) studied KG extraction under domain-range constraints, here we additionally consider qualifiers constraints, which we formalize next.

Qualifier constraints. Each predicate $r \in \mathcal{R}$ specifies a set $\mathsf{qual}(r) \subseteq \mathcal{Q} \subset \mathcal{R}$ of admissible qualifier predicates. For example, the qualifier $(\mathsf{point\ in\ time}, 1938)$ from above is consistent with $\mathsf{educated\ at}$ if $\mathsf{point\ in\ time} \in \mathsf{qual}(\mathsf{educated\ at})$, whereas $(\mathsf{color}, \mathsf{red})$ is inconsistent if $\mathsf{color} \notin \mathsf{qual}(\mathsf{educated\ at})$. Furthermore, a qualifier (r, o) satisfies the range constraint of r if $t_o \prec u_o$ for some $t_o \in \tau(o), u_o \in \mathsf{rng}(r)$.

Symbolic querying via SPARQL. The ontology constraints defined above provide the *rules* necessary to write SQL-like queries for symbolic retrieval or QA, using SPARQL (Harris et al., 2013). For example, consider the task of retrieving *all physicists who received the Nobel prize after 2000*. This can be expressed as the following query.

```
1 SELECT ?person WHERE {
2     ?person instanceOf: Human: .
3     ?person occupation: Physicist: .
4     ?person awardReceived: ?stmt .
5     ?stmt awardValue: NobelPrizePhysics: ;
6           pointInTime: ?date .
7     FILTER ( ?date >= "2000-01-01" ) }
```

In the above, L2 specifies a type constraint over the entity variable `?person`, while in L3 the range constraint of `occupation:` tells us to put the occupation `Physicist:` on the right-hand side of the triple. The fact that `pointInTime:` is an allowed qualifier by `awardReceived:` guarantees we can use it to access the additional time information of the award relationship, as shown in L5-L7 using statement expressions following the Wikidata data model (Vrandečić and Krötzsch, 2014).

In Section 3 we propose a method for KG extraction that maps open-domain types and predicates to the ones allowed by the ontology. Then, in Section 4 we develop an approach to systematically correct violations of ontology constraints.

3 KG Extraction and Canonicalization

Given a source text $\mathcal{S}$, we aim at extracting a KG $\mathcal{G}$ equipped with typed entities and qualifiers such that

it encodes the relationships between entities in $\mathcal{S}$. Here, we present a method that (1) performs open-domain KG construction, and (2) maps (or canonicalizes) the extracted open-domain types (resp. predicates) to the types (resp. predicates) allowed by the ontology, i.e., $\mathcal{T}$ (resp. $\mathcal{R}$). To make (2) efficient, we leverage embeddings computed on types and predicates labels, while we use LLMs for disambiguation only. We dub this method as *ontology-aligned KG construction* (OAK), and defer the problem of satisfying the ontology constraints to Section 4. In Appendix G we report all our prompts.

3.1 Open-domain Extraction

Step 1. We use a single prompt instructing an LLM to extract triples, entity types, and optionally a number of qualifiers for each triple from the source text $\mathcal{S}$. All extracted elements are open-domain: the ontology is not part of the input context, and the extracted types and predicates are instead mapped to it in subsequent steps (see Section 3.2). This KG extraction step differs from the one in Chepurova et al. (2026) by the fact that we also extract the types of qualifier objects, as required by the constraints over qualifiers (see Section 2). We use a fixed in-context example to improve the adherence to the required JSON output format.

Open-domain KG extraction avoids having to first identify which ontology fragments are relevant to the input text and include them in the LLM prompt, as done in Zhang and Soh (2024); Feng et al. (2024). As a result, the prompt is shorter, reducing token usage and thereby improving prefix cache hit rates as well. In addition, in our experiments using a reasoning LLM, we find that providing the ontology constraints as part of the input context drastically reduces the number of extracted triples (Section 5.1), which in turn reduces QA performances as well (Section 5.2). Perhaps surprising, this represents evidence that leveraging constraints at KG extraction time can sometimes negatively affect KG construction and QA.

3.2 Canonicalization of Types and Predicates

Step 2. We map (or canonicalize) the open-domain types and predicates extracted in **Step 1** to types and predicates allowed by the ontology, respectively. Recently, Chepurova et al. (2026) proposed to *perform multiple LLM calls for each extracted triple* with the purpose of canonicalizing the subject and object entity types, followed by canonicalizing the predicate such that the resulting triple satisfies the domain-range constraint. However, this accounts for a significant tokens overhead when applied to several thousands triples.

We seek a more token efficient approach by making the following simplifying assumptions. First, we assume that entities that have been associated to the same open-domain type in **Step 1** should have the same canonical type from the ontology. Second, we assume that extracted triples sharing the same open-domain predicate express the same relationship semantics, and therefore their predicates should be canonicalized in the same way. These simplifying assumptions are not expected to hold universally, and their purpose is to minimize the number of LLM calls at the cost of introducing schema violations. Nevertheless, these violations can be later resolved in our framework based on post-extraction corrections (see Section 4). We exploit the assumptions above in a canonicalization step leveraging embedding similarities.

Formally, let $\hat{t}$ be a type label that has been extracted in **Step 1**, which we aim to map to some $t \in \mathcal{T}$. We compute the text embedding similarity between the extracted $\hat{t}$ and the label of each $t \in \mathcal{T}$, and pick the type having the highest similarity score. Following Chepurova et al. (2026) we also consider any available label aliases for each t. However, multiple types might have very similar labels or even have common aliases, making canonicalization through embedding similarities alone insufficient. Whenever multiple types have close similarity scores w.r.t. $\hat{t}$ based on a hyperparameter β, we prompt an LLM to select the most suitable one. We construct the set of candidates types as

$$\mathcal{C}_{\mathcal{T}}(\hat{t}) = \{t_i \in \mathcal{T} \mid \phi(\hat{t}, t_i) \geq m_{\hat{t}} - \beta\}$$

where $m_{\hat{t}} = \max_{t_i \in \mathcal{T}}\{\phi(\hat{t}, t_i)\}$ and $\phi(\hat{t}, t_i)$ denotes the maximum embedding similarity between $\hat{t}$ and any label or alias of t_i. Then, the canonical t for $\hat{t}$ is chosen by calling an LLM, i.e., $t = \text{LLM}(\mathcal{S}, \hat{t}, \mathcal{C}_{\mathcal{T}}(\hat{t}), E_{\hat{t}})$, where $E_{\hat{t}}$ denotes the set of entities that have been previously typed with $\hat{t}$. In the particular case of $|\mathcal{C}_{\mathcal{T}}(\hat{t})| = 1$, no LLM calls are made and thus β can be used to tune the tokens cost. In our experiments we use the cosine similarity and empirically set $\beta = 0.05$.

We proceed similarly to canonicalize the predicates. Given an open-domain predicate label $\hat{r}$ extracted in **Step 1**, we build the set of candidates

$$\mathcal{C}_{\mathcal{R}}(\hat{r}) = \{r_i \in \mathcal{R} \mid \phi(\hat{r}, r_i) \geq m_{\hat{r}} - \beta\},$$

where $m_{\hat{r}} = \max_{r_i \in \mathcal{R}}\{\phi(\hat{r}, r_i)\}$. Then, the canonical $r \in \mathcal{R}$ for $\hat{r}$ is chosen by calling an LLM, i.e., $r = \text{LLM}(\mathcal{S}, \hat{r}, \mathcal{C}_{\mathcal{R}}(\hat{r}), T_{\hat{r}})$, where $T_{\hat{r}} \subset \mathcal{E} \times \mathcal{E}$ denotes the subject-object entity pairs that have been linked with the open-domain predicate $\hat{r}$ in **Step 1**.

Finally, since qualifier predicates specify additional information about the triples themselves, their number is typical much smaller than the total number of predicates (e.g., it is only ~29% in the Wikidata ontology fragment we consider in our experiments). This reduces ambiguity during canonicalization, allowing us to simply map each qualifier predicate name to the higher scoring qualifier predicate in the ontology. Finally, we remove duplicate entitites as described next.

3.3 Types-Aware Entity Deduplication

Step 3. A common approach to extracting a KG from a corpus consists of splitting the texts into chunks, running the KG extraction method on each chunk and then merging the obtained KGs (Zhang and Soh, 2024; Mo et al., 2025). However, this often yields entities with different labels that refer to the same concept, making an entity deduplication step necessary. Recently, Mo et al. (2025) proposed a two-step algorithm that first clusters entities based on their embeddings, and then uses an LLM to iteratively identify duplicates within each cluster. We adopt this method while also exploiting the type information from **Step 2**. In particular, entities typed as literals (e.g., a number or a calendar date) are excluded from deduplication, hence reducing the clusters size.

Until now, we discussed OAK: a method to extract a KG such that entity types and predicates are mapped to the ones in a given ontology. However, the extracted KG might still contain ontology constraint violations, which we aim to correct next.

4 Correction of Ontology Violations

The entity types allow us to symbolically detect which triples and qualifiers violate which ontology constraints. This contrasts with methods that rely on LLMs for error detection, such as detecting disallowed types (Arun et al., 2025), classifying class membership errors (Allen and Groth, 2024), and supporting human-in-the-loop validation (Regino and dos Reis, 2025).

After we collect the triples and qualifiers violating the ontology, we prompt an LLM to correct them by choosing from a set of syntactic transformations. Note that, since we correct ontology violations *after* the KG has been extracted, our method can be applied *ex-post* to any KG extraction approach, Next, we describe our approach to correct inconsistent triples.

4.1 Correcting Triples

To motivate the set of syntactic transformations useful to correct ontology violations, we identify a number of KG extraction mistakes – each of which suggesting a particular corrective action.

V1. The **subject-object inversion** error occurs when the types of the subject s (resp. object o) in a triple (s, r, o) are inconsistent with the domain (resp. range) of the predicate r because the triple direction is reversed, while the inverted triple (o, r, s) is instead consistent. For example, if $\mathsf{dom}(\mathsf{author}) = \{\mathsf{book}\}$ and $\mathsf{rng}(\mathsf{author}) = \{\mathsf{human}\}$, then the triple $(\mathsf{G.\ Orwell}, \mathsf{author}, \mathsf{Animal\ Farm})$ is inconsistent when $\tau(\mathsf{G.\ Orwell}) = \{\mathsf{human}\}$ and $\tau(\mathsf{Animal\ Farm}) = \{\mathsf{book}\}$. Although this triple expresses that G. Orwell is the author of Animal Farm, it violates the ontology because the subject and object are reversed. It can therefore be corrected by *swapping subject and object*. We find this mistake to be frequent, and thus automatically apply the correction whenever possible.

V2. The error of **predicate mis-canonicalization** occurs when a predicate has been wrongly selected: the triple (s, r, o) is inconsistent because r does not explain a relationship between entities s and o. For example, the triple $(\mathsf{Animal\ Farm}, \mathsf{distribution\ format}, \mathsf{Penguin\ Books})$ might have been hallucinated because the publisher Penguin Books has "Books" in its label. This triple is inconsistent because the type organization of Penguin Books is not allowed by the range constraint of distribution format. Instead, $(\mathsf{Animal\ Farm}, \mathsf{publisher}, \mathsf{Penguin\ Books})$ satisfies the domain-range constraint of publisher. Thus, the corrective action is to *replace the predicate*.

V3. Lastly, a triple (s, r, o) may be inconsistent because of **underspecified or missing types**: the types of the subject s (resp. the object o) are not specific enough to satisfy the domain (resp. the range) of r or, alternatively, a type might be missing. For example, assuming that $\tau(\mathsf{Animal\ Farm}) = \{\mathsf{written\ work}\}$, then $(\mathsf{Animal\ Farm}, \mathsf{author}, \mathsf{G.\ Orwell})$ is inconsistent with the domain of author, as book $\prec$ written work. That is, the type written work is not

Dataset: HotpotQA	Triples Statistics		Qualifiers Statistics		# Tokens ($\times 10^6$)		
	# Total	% Valid	# Total	% Valid	Prompt	Completion	Weighted
QWEN3-30B-A3B							
KGGEN	155 391 ± 185	—	0	—	71.7	11.0	28.9 (0.64)
WIKONTIC	117 891 ± 79	77.4 ± 0.2	45 438 ± 455	—	139.3	10.5	45.3 (1.00)
IN-CONTEXT	173 942 ± 420	79.4 ± 0.2	70 763 ± 352	82.6 ± 0.2	93.3	12.5	35.8 (0.79)
OAK	127 763 ± 107	63.4 ± 0.1	45 721 ± 516	33.6 ± 0.1	34.3	7.7	**16.3** (0.36)
OAK+MEND	127 108 ± 130	**96.8 ± 0.1**	45 721 ± 516	**91.5 ± 0.1**	74.0	8.3	26.8 (0.59)
GPT-OSS-120B							
KGGEN	128 163 ± 4440	—	0	—	87.8	54.8	76.8 (0.88)
WIKONTIC	110 109 ± 329	**99.0 ± 0.1**	40 939 ± 35	—	139.5	52.4	87.3 (1.00)
IN-CONTEXT	67 197 ± 144	97.9 ± 0.1	8639 ± 455	**97.7 ± 0.1**	98.9	21.5	46.2 (0.53)
OAK	113 843 ± 237	67.9 ± 0.3	45 351 ± 26	35.7 ± 0.3	33.8	28.4	**36.8** (0.42)
OAK+MEND	112 737 ± 1093	98.4 ± 1.0	45 176 ± 273	96.8 ± 1.7	67.7	52.3	69.2 (0.79)

Table 1: **KG Construction with OAK+MEND achieves high ontology consistency rates for both triples and qualifiers, yet requires modest token cost.** We show the number of triples and qualifiers extracted by different methods for KG construction, as well as the percentage consistent with ontology constraints (see Section 2), where applicable. We also show the total number of prompt and completion tokens (in millions), and a weighted cost combination that takes into account a prompt-to-completion cost ratio of 1/4 (e.g., as the $ price ratio for GPT-4.1).

specific enough as required by the predicate author. Here, the correction is *adding a type to the subject*.

The cases **V1-3** above motivate prompting an LLM to choose among the corrective actions:

- swap: swap the subject s and object o entities.
- replace_predicate: substitute r with $r' \in \mathcal{R}$ such that (s, r', o) is consistent. The candidates r' are ranked based on the embedding similarity w.r.t. the label of r.
- add_subject_type: add a new type to s, where the candidate types are in $\mathsf{dom}(r)$.
- add_object_type: add a new type to o, where the candidate types are in $\mathsf{rng}(r)$.

Since multiple actions might be needed to fix a domain-range violation (e.g., adding a type followed by swapping), the LLM is prompted to return a composition of actions. Moreover, by allowing the LLM to add entity types and by propagating the updated types to all triples, *our method can effectively correct multiple constraint violations with a single LLM call*, making it particularly efficient.

4.2 Correcting Qualifiers

To correct qualifiers violating the ontology, we proceed similarly to the triples corrections above. As detailed in Appendix C, qualifier constraints indicate two main sources of errors: using a disallowed qualifier predicate, or the types of the qualifier object are underspecified/missing (see Section 2). Given an inconsistent qualifier (r_q, o_q) associated to a triple (s, r, o), we prompt an LLM to choose one corrective action among:

- replace_predicate: substitute r_q with $r'_q \in \mathsf{qual}(r)$ such that (r'_q, o_q) is consistent. The candidates r'_q are ranked based on the embedding similarity w.r.t. the label of r_q.
- add_object_type: add a new type to o_q, where the candidate types are in $\mathsf{rng}(r_q)$.

In all correction prompts we insert a one-line explanation of why a triple or qualifier violates a constraint. To keep corrections grounded in the source text, we also add the text fragment from which the inconsistent triple or qualifier was originally extracted from. Finally, we give the option to *not* choose any action whenever all possible corrections would yield a fact unsupported by the text.

4.3 Corrective Actions Re-validation

Each corrective action is followed by ontology-based re-validation. Once a triple or qualifier is deemed consistent with the ontology, it is retained and excluded from subsequent corrections, preventing the introduction of new ontology violations. MEND performs up to two correction iterations, as some facts require sequential actions (e.g., adding a type and then swapping the entities). The remaining invalid facts correspond to cases in which the LLM fails to identify an action that produces an ontology-valid result while preserving consistency with the input text. We observed no significant improvement in the considered metrics when increasing the number of iterations beyond two.

5 Empirical Evaluation

We answer the following questions: **Q1.** How consistent are the KGs extracted by our method w.r.t.

Method	HotpotQA				MuSiQue			
	QWEN3-30B-A3B		GPT-OSS-120B		QWEN3-30B-A3B		GPT-OSS-120B	
	EM	F_1	EM	F_1	EM	F_1	EM	F_1
ZEROSHOT	21.8 ± 0.3	30.7 ± 0.3	33.4 ± 0.7	45.1 ± 0.5	2.3 ± 0.1	9.4 ± 0.7	12.9 ± 0.7	24.1 ± 0.2
VECTORRAG	42.9 ± 0.4	56.5 ± 0.3	58.6 ± 0.2	72.9 ± 0.1	14.0 ± 0.4	23.2 ± 0.5	30.4 ± 0.7	43.6 ± 0.8
HIPPORAGV2	50.0 ± 0.8	64.1 ± 0.6	38.8 ± 0.7	57.0 ± 0.5	22.0 ± 0.1	33.9 ± 1.0	19.4 ± 0.1	36.2 ± 0.2
KGGEN	38.5 ± 1.5	48.7 ± 1.6	38.9 ± 0.5	48.5 ± 1.3	19.2 ± 0.2	29.2 ± 0.2	21.9 ± 0.6	28.6 ± 0.6
WIKONTIC	46.4 ± 0.7	57.7 ± 0.5	40.4 ± 0.8	50.4 ± 0.8	**25.0 ± 1.0**	**37.4 ± 0.1**	19.9 ± 0.2	28.0 ± 0.6
IN-CONTEXT	45.0 ± 0.7	55.6 ± 0.9	33.1 ± 0.7	41.8 ± 0.8	22.0 ± 0.7	34.0 ± 0.8	17.5 ± 0.5	23.7 ± 0.7
OAK	**47.0 ± 1.0**	**58.6 ± 0.7**	**46.4 ± 0.9**	**57.8 ± 1.2**	24.4 ± 0.7	36.5 ± 0.9	**22.6 ± 1.3**	**31.1 ± 0.7**
OAK+MEND	45.6 ± 0.4	56.4 ± 0.5	46.1 ± 2.0	56.8 ± 1.7	23.3 ± 1.1	35.2 ± 1.4	21.5 ± 0.1	29.9 ± 0.2

Table 2: **OAK achieves competitive QA results, and our corrections enhancing ontology consistency account for little or no loss of performances.** We show QA performances in terms of exact match (EM) and F_1 metrics computed on predicted and ground-truth answers, achieved by leveraging the multi-step QA method originally developed in Chepurova et al. (2026). For HIPPORAGV2 we report QA results achieved by running their code.

an ontology, and how token efficient it is? **Q2.** Does our algorithm correcting ontology violations still preserve the text semantics as required by the QA task? **Q3.** Are the extracted KGs well suited for symbolic querying via SPARQL queries?

Datasets and ontology. Following Gutierrez et al. (2024); Chepurova et al. (2026), we focus on fragments of HotpotQA (Yang et al., 2018) and MuSiQue (Trivedi et al., 2022) datasets consisting of text paragraphs and questions in natural language. Following Feng et al. (2024); Chepurova et al. (2026), we also use a Wikidata ontology fragment, consisting of a types hierarchy, allowed predicates, and the related constraints (Vrandečić and Krötzsch, 2014). We detail these in Appendix A.

Baselines. We compare against: KGGEN, an ontology-free KG extraction method (Mo et al., 2025); WIKONTIC targeting the Wikidata type hierarchy and domain-range constraints (Chepurova et al., 2026); and a custom-built in-context baseline inserting a verbalization of the constraints defined in Section 2 as part of the LLM context (details in Appendix D). Although KGGEN does not rely on an existing ontology, we still compare against it in terms of efficiency and downstream QA performances. To the best of our knowledge, WIKONTIC is the closest to our method with an open implementation. Following the QA setting adopted by WIKONTIC, we retain triples and qualifiers violating the ontology. We also report results of QA methods retrieving source text fragments and using them as part of the LLM context when answering questions, such as vector RAG (Lewis et al., 2020) and HippoRAGv2 (Gutiérrez et al., 2025).

LLMs. All our experiments use either the instruction-tuned QWEN3-30B-A3B-INSTRUCT-2507-FP8 (QWEN3-30B-A3B for brevity) or the reasoning model GPT-OSS-120B, together with QWEN3-EMBEDDING-0.6B for text embeddings.

5.1 Q1. Consistency and Efficiency

In Table 1 we show the number of triples and qualifiers extracted on HotpotQA, as well as the fraction of triples and qualifiers that are consistent. By applying our post-extraction corrections (Section 4), dubbed as OAK+MEND, we improve the ontology consistency of both triples and qualifiers extracted by OAK (Section 3) i.e., we increase the percentage of valid triples (resp. qualifiers) from 63.4% (resp. 33.6%) to 96.8% (resp. 91.5%) with QWEN3-30B-A3B; and from 67.9% (resp. 35.7%) to 98.4% (resp. 96.8%) with GPT-OSS-120B. When using QWEN3-30B-A3B, the achieved ontology consistency is higher than the IN-CONTEXT baseline and WIKONTIC, while still comparable when using GPT-OSS-120B. Yet our OAK+MEND method is more efficient in terms of prompt and completion tokens. We show this by weighting prompt and completion tokens differently given their different overhead. When compared to the token cost of WIKONTIC (normalized 100%), the cost of OAK+MEND is 59% on QWEN3-30B-A3B; and 79% on GPT-OSS-120B. Table F.1 shows similar results on MuSiQue.

As a minor observation, we note a slight decrease in the number of triples from OAK to OAK+MEND in Table 1. This decrease is due to deduplication: because triples are stored as a set, correcting an inconsistent triple may make it identical to an existing one, causing the duplicate to be automatically removed. Furthermore, Table 3 reports the frequency of each triple corrective action kind as predicted by the LLMs on HotpotQA,

showing that predicate replacement is the action predicted with the highest frequency.

Instead, we do not report a "no action" rate because "no action" can result from several situations, making this rate difficult to compare meaningfully. For example, cases in which the LLM selects a replacement predicate outside the candidate list are also considered "no action." Nevertheless, given MEND's near-100% consistency rate, such cases are expected to be relatively small when compared to the reported action frequencies.

	QWEN3-30B-A3B	GPT-OSS-120B
Triples		
`swap`*	9671 ± 35	8520 ± 1175
`add_subject_type`	1119 ± 30	3148 ± 775
`add_object_type`	694 ± 53	5784 ± 459
`replace_predicate`	45 539 ± 30	34 715 ± 503
Qualifiers		
`add_object_type`	1678 ± 9	2132 ± 354
`replace_predicate`	11 581 ± 228	12 777 ± 259

Table 3: **Predicate replacement and subject-object swap are the most common actions.** We report the frequency of predicted corrective actions for triples and qualifiers. When using QWEN3-30B-A3B (resp. GPT-OSS-120B), the number of triples whose either subject or object types have been updated is 12058.7 ± 246.4 (resp. 27408.7 ± 571.8), hence showing that the impact of subject and object types addition on the triples corrections is still lower than the replacement of predicates. **Swap* statistics include both modalities: LLM-predicted swaps and automatic swaps when they yield a valid triple.

Qualitative evaluation. Our framework alters the extracted entity types and triples, which can in turn modify the encoded semantics. In order to rule out whether OAK+MEND fixes the ontology violations by sacrificing semantic faithfulness, Appendix E reports a human audit measuring the alignment of extracted triples and entity types w.r.t. the source text *before* and *after* corrections. After the triples corrections, Table E.1 shows an increase in triples faithfulness and no loss in types faithfulness on HotpotQA. Although brief, our analysis suggests that the ontology consistency improvements shown in Table 1 do not come at the cost of a lower text fidelity. In the next section, this claim is further validated quantitatively by evaluating OAK+MEND on downstream QA.

5.2 Q2. Question Answering Experiments

We evaluate the capability of our method in encoding knowledge useful for QA. For all methods we adopt the multi-step QA approach designed in Chepurova et al. (2026), which iteratively decomposes a multi-hop question into simpler 1-hop ones each being answered with an LLM. When answering to a 1-hop question, the LLM is asked to extract relevant entities from the question, which are then linked to entities in the extracted KG via embedding similarity. The linked entities are used to retrieve the associated triples and qualifiers, which provide the context for answering the question. Importantly, for each question, we do not keep separate KGs for its supporting texts; instead, we merge the knowledge extracted from all supporting texts into a single unified KG before performing QA.

Following the literature on QA, in Table 2 we report exact match (EM) and F_1 metrics computed between predicted and ground-truth answers, excluding articles. OAK achieves the best or second-best QA results, thus being competitive with KGGEN, WIKONTIC and the IN-CONTEXT baseline. Moreover, applying ontology-based corrections causes little or no loss in QA performance, suggesting that the corrections preserve the semantics of the extracted triples and qualifiers. In Table 2 we also report QA performances achieved by text-based methods. That is, vector RAG and HippoRAGv2 answer questions by directly retrieving and reasoning over the original source text rather than operating exclusively on symbolic triples and their qualifiers like OAK. While we observe that operating directly on the text can boost QA performances on some configurations (e.g., vector RAG with GPT-OSS-120B on HotpotQA), these methods do not support symbolic reasoning. Finally, Table F.2 shows an ablation on the qualifiers, further supporting the finding of Chepurova et al. (2026) that qualifiers boost QA performances.

5.3 Q3. Patterns Enabling SPARQL Queries

We evaluate how amenable the extracted KGs are to SPARQL queries, for which consistency with the underlying ontology is crucial (see Section 2). To disentangle our evaluation with the challenging task of translating questions in natural language to SPARQL (Gashkov et al., 2025; Perevalov and Both, 2025), we introduce a benchmark measuring the occurrence of *basic graph patterns* (BGPs). BGPs are abstract sub-graphs defined as a conjunction of triples forming the `WHERE` clause in a SPARQL query (e.g., the query in Section 2). Hence, their presence in the extracted KGs is a necessary condition for executing SPARQL queries.

BGPs datasets. We build a dataset of BGPs of

Method	HotpotQA				MuSiQue			
	Avg. edge multiplicity	$\%(s,o)$ w/ ≥ 2 links	h-index	i100-index	Avg. edge multiplicity	$\%(s,o)$ w/ ≥ 2 links	h-index	i100-index
QWEN3-30B-A3B								
WIKONTIC	1.05	4.3	339.7 ± 0.6	895.3 ± 10.8	1.11	8.6	341.7 ± 3.5	977.0 ± 3.6
IN-CONTEXT	(1.18)	(14.1)	**515.7 ± 5.0**	**1821.0 ± 19.1**	(1.26)	(17.6)	**643.0 ± 2.6**	**2491.7 ± 35.7**
OAK	1.04	3.2	336.0 ± 1.0	884.7 ± 4.2	1.10	7.8	364.0 ± 1.7	1031.7 ± 6.8
OAK+MEND	1.03	2.7	402.7 ± 2.5	1141.7 ± 16.8	1.08	6.6	430.3 ± 2.5	1347.0 ± 4.6
GPT-OSS-120B								
WIKONTIC	1.05	4.4	370.3 ± 3.1	**1068.0 ± 8.5**	1.11	9.0	401.7 ± 3.1	**1182.0 ± 3.0**
IN-CONTEXT	1.02	2.0	320.7 ± 2.1	775.3 ± 8.0	1.05	4.7	337.3 ± 6.7	902.0 ± 7.9
OAK	1.04	3.2	329.0 ± 3.5	823.3 ± 16.5	1.13	10.1	344.3 ± 1.2	901.7 ± 5.8
OAK+MEND	1.03	2.6	**383.3 ± 6.7**	**1054.7 ± 25.5**	1.10	8.1	**408.7 ± 3.5**	1157.0 ± 9.2

Table 4: **OAK+MEND extracts KGs containing graph patterns useful to construct and run SPARQL queries.** For each method we show our h-index and i100-index metrics computed on the patterns frequencies, as well as edge multiplicity metrics measuring the ambiguity in choosing the right predicate between two entities (see Section 5.3). On QWEN3-30B-A3B, the IN-CONTEXT method achieves an outlier edge multiplicity metrics (see circled numbers), thus boosting the number of pattern matches by extracting redundant-triples (see also Table F.6). Table F.5 complements these results with i1-index and i5-index metrics, which reward BGP diversity more when compared to h-index and i100-index, showing OAK+MEND being competitive in terms of graph patterns diversity.

queries executed on the Wikidata endpoint, which we extract from the LSQ-2.0 collection (Saleem et al., 2015; Stadler et al., 2024). This yields a set of a few thousands BGPs over hundreds of predicates. Moreover, to obtain a larger set of patterns, we artificially generate BGPs from the ontology domain-range constraints using templates common in the complex query answering literature (Gregucci et al., 2025). We detail the procedures and the statistics of the two BGPs datasets in Appendix B.

Metrics. One way to evaluate the extracted KGs would be to simply compare the total frequency of BGP matches. However, since some BGPs appear much more frequently than others, we use metrics that also account for *rare* BGPs. Inspired by the h-index and i10-index metrics, which reward a consistent output of highly cited papers, we adopt analogous metrics to reward methods that match many BGPs with consistently high frequency across a *diverse set of* patterns. In Table 4 we report h-index and i100-index metrics computed on all patterns, together with the average number of links connecting two entities (edge multiplicity), and the the percentage of connected subject-object pairs with two or more links. The last two metrics serve as a sanity check for redundant triples that encode the same information with different predicates, since such redundancy could artificially boost BGPs matches at the cost of introducing ambiguity.

Results. Our OAK+MEND on QWEN3-30B-A3B is second only to the IN-CONTEXT baseline, which however has high edge multiplicity. As we detail in Table F.6, this is because IN-CONTEXT extracts highly-redundant triples over pairs of entities encoding the same information, yet with different predicates that have slightly different meaning. Instead, on GPT-OSS-120B, our OAK+MEND achieves similar performances to WIKONTIC, and much better than IN-CONTEXT, without having an outlier edge multiplicity. In all cases our ontology-based corrections boost h-index. In Table F.3 and Table F.4 we report the results achieved on only the LSQ-2.0-based or artificial patterns, respectively.

6 Conclusion

We introduced a method for ontology-grounded KG extraction that corrects ontology violations *ex-post*, improving token efficiency compared to existing methods based on repeated LLM calls or long ontology-aware contexts. Our approach is supported by experiments evaluating the many facets of KG extraction: ontology consistency, QA performances, token efficiency, and suitability for symbolic querying. In particular, we have shown that targeted LLM-predicted corrections substantially improve ontology consistency while preserving semantic accuracy. To the best of our knowledge, this is the first work to study KG extraction through such a broad empirical evaluation, while also providing an in-depth evaluation over the Wikidata ontology. Finally, we believe our SPARQL graph-patterns benchmark can serve as a useful tool for quantitatively assessing how well a KG extraction method supports SPARQL-based querying.

Limitations

Our experiments focused on the general-purpose ontology available in Wikidata, as this is a popular setting in the KG extraction community. Instead, one can consider custom-build ontologies that may be specific for a certain domain, e.g., for biomedical KGs (Walsh et al., 2020). We believe understanding whether the performances achieved by LLMs generalize to custom or even closed-source ontologies remains poorly understood. In addition, our method requires a detailed ontology specification, including both domain-range and qualifiers constraints, which might not be available.

Moreover, in our experiments we restricted ourselves to open-weight LLMs only. While this promotes transparency and reproducibility over time, further experiments exploiting closed-weight yet more powerful LLMs could provide a more detailed overview of the performances achieved by the different KG extraction methods.

The lack of an ontology-aligned ground-truth KG for the considered benchmarks does not allow us to evaluate our simplifying assumptions for the efficient canonicalization of types and predicates (Section 3.2). Nevertheless, our qualitative analysis in Appendix E partially addresses this by showing that the canonicalized entity types are faithful w.r.t. the source text in 80-85% of the analyzed triples.

Lastly, our experiments do not show QA performances achieved by means of SPARQL queries executed on the extracted KGs. This is because it would require leveraging text-to-SPARQL translation techniques to convert questions in natural language to SPARQL queries (Gashkov et al., 2025; Perevalov and Both, 2025). Since this is already a very challenging open problem, in our work we instead rely on proxy metrics based on the occurrences of SPARQL query graph patterns alone.

Acknowledgements

We would like to acknowledge Antonio Vergari for meaningful discussions during the early stages of this work. We also thank the anonymous reviewers for their constructive feedback during the rebuttal.

References

Shqiponja Ahmetaj, Robert David, Axel Polleres, and Mantas Simkus. 2022. Repairing shacl constraint violations using answer set programming. In *International Workshop on the Semantic Web*.

Bradley Paul Allen and Paul T. Groth. 2024. Evaluating class membership relations in knowledge graphs using large language models. In *ESWC Satellite Events*.

Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. 2021. Complex query answering with neural link predictors. In *ICLR*. OpenReview.net.

Abhinav Arun, Fabrizio Dimino, Tejas Prakash Agarwal, Bhaskarjit Sarmah, and Stefano Pasquali. 2025. FinReflectKG: Agentic construction and evaluation of financial knowledge graphs. In *Proceedings of the 6th ACM International Conference on AI in Finance*, ICAIF 2025.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *ACL*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. FinQA: A dataset of numerical reasoning over financial data. In *EMNLP*.

Alla Chepurova, Aydar Bulatov, Mikhail Burtsev, and Yuri Kuratov. 2026. Wikontic: Constructing Wikidata-aligned, ontology-aware knowledge graphs with large language models. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Shaohan Wang, Pengyu Wang, Xiaorui Wang, and Zhendong Mao. 2026. A-rag: Scaling agentic retrieval-augmented generation via hierarchical retrieval interfaces. *Preprint*, arXiv:2602.03442.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2025. From local to global: A graph rag approach to query-focused summarization. *Preprint*, arXiv:2404.16130.

Xiaohan Feng, Xixin Wu, and Helen Meng. 2024. Ontology-grounded automatic knowledge graph construction by llm under wikidata schema. *Preprint*, arXiv:2412.20942.

Nicolas Ferranti, Jairo Francisco De Souza, Shqiponja Ahmetaj, and Axel Polleres. 2024. Formalizing and validating wikidata's property constraints using SHACL and SPARQL. *Semantic Web*, 15(6):2333–2380.

Mikhail Galkin, Jincheng Zhou, Bruno Ribeiro, Jian Tang, and Zhaocheng Zhu. 2024. A foundation model for zero-shot logical query reasoning. *NeurIPS*.

Hang Gao and Dimitris N. Metaxas. 2026. Beyond explicit edges: Robust reasoning over noisy and sparse knowledge graphs. *Preprint*, arXiv:2603.14006.

Aleksandr Gashkov, Aleksandr Perevalov, Maria Eltsova, and Andreas Both. 2025. SPARQL query generation with llms: Measuring the impact of training data memorization and knowledge injection. In *ICWE*.

Cosimo Gregucci, Bo Xiong, Daniel Hernández, Lorenzo Loconte, Pasquale Minervini, Steffen Staab, and Antonio Vergari. 2025. Is complex query answering really complex? In *Forty-second International Conference on Machine Learning*.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. In *NeurIPS*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From RAG to memory: Non-parametric continual learning for large language models. In *ICML*.

Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. 2013. SPARQL 1.1 query language.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*.

Tomaz Hocevar and Janez Demsar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4):559–565.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. 2018. Learning deep generative models of graphs. *CoRR*, abs/1803.03324.

Tung-Wei Lin, Gabe Fierro, Han Li, Tianzhen Hong, Pierluigi Nuzzo, and Alberto Sangiovanni-Vincentelli. 2025. Systematic evaluation of knowledge graph repair with large language models. *ArXiv*, abs/2507.22419.

Yuxing Lu, Wei Wu, Xukai Zhao, Rui Peng, and Jinzhuo Wang. 2025. KARMA: Leveraging multi-agent LLMs for automated knowledge graph enrichment. In *NeurIPS*.

Nandana Mihindukulasooriya, Sanju Tiwari, Carlos F. Enguix, and Kusum Lata. 2023. Text2kgbench: A benchmark for ontology-driven knowledge graph generation from text. In *ISWC*, volume 14266 of *Lecture Notes in Computer Science*, pages 247–265. Springer.

Belinda Mo, Kyssen Yu, Joshua Kazdan, Proud Mpala, Lisa Yu, Chris Cundy, Charilaos I. Kanatsoulis, and Sanmi Koyejo. 2025. KGGen: Extracting knowledge graphs from plain text with language models. In *NeurIPS*.

Jixuan Nie, Xia Hou, Wenfeng Song, Xuan Wang, Xingliang Jin, Xinyu Zhang, ShuoZhe Zhang, and Jiaqi Shi. 2024. Knowledge graph efficient construction: Embedding chain-of-thought into llms. In *VLDB Workshops*. VLDB.org.

Jeff Z. Pan, Simon Razniewski, Jan-Christoph Kalo, Sneha Singhania, Jiaoyan Chen, Stefan Dietze, Hajira Jabeen, Janna Omeliyanenko, Wen Zhang, Matteo Lissandrini, Russa Biswas, Gerard de Melo, Angela Bonifati, Edlira Vakaj, Mauro Dragoni, and Damien Graux. 2023. Large Language Models and Knowledge Graphs: Opportunities and Challenges. *Transactions on Graph Data and Knowledge*, pages 2:1–2:38.

Ciyuan Peng, Feng Xia, Mehdi Naseriparsa, and Francesco Osborne. 2023. Knowledge graphs: Opportunities and challenges. *Artificial Intelligence Review*, pages 1 – 32.

Aleksandr Perevalov and Andreas Both. 2025. Text-to-sparql goes beyond english: Multilingual question answering over knowledge graphs through human-inspired reasoning. *CoRR*, abs/2507.16971.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Conference on Empirical Methods in Natural Language Processing*.

André Gomes Regino and Julio Cesar dos Reis. 2025. Can LLMs be knowledge graph curators for validating triple insertions? In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, pages 87–99, Abu Dhabi, UAE. International Committee on Computational Linguistics.

Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *ICLR*. OpenReview.net.

Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. LSQ: the linked SPARQL queries dataset. In *ISWC (2)*, volume 9367 of *Lecture Notes in Computer Science*, pages 261–269. Springer.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: recursive abstractive processing for tree-organized retrieval. In *ICLR*. OpenReview.net.

Claus Stadler, Muhammad Saleem, Qaiser Mehmood, Carlos Buil-Aranda, Michel Dumontier, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. 2024. LSQ 2.0: A linked dataset of SPARQL query logs. *Semantic Web*, 15(1):167–189.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Zeno van Cauter and Nikolay Yakovets. 2024. Ontology-guided knowledge graph construction from maintenance short texts. In *Proceedings of the 1st Workshop on Knowledge Graphs and Large Language Models (KaLLM 2024)*. Association for Computational Linguistics.

Ellen M. Voorhees and Dawn M. Tice. 2000. The trec-8 question answering track. *Proceedings of the Language Resources and Evaluation Conference*.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Brian Walsh, Sameh K. Mohamed, and Vít Nováček. 2020. BioKG: A Knowledge Graph for Relational Learning On Biological Data. In *CKIM*.

Dali Wang and Mizuho Iwaihara. 2025. OSKGC: A benchmark for ontology schema-based knowledge graph construction from text. In *KBC-LM and LM-KBC at ISWC*.

Orion Weller, Michael Boratko, Iftekhar Naim, and Jinhyuk Lee. 2026. On the theoretical limitations of embedding-based retrieval. In *The Fourteenth International Conference on Learning Representations*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*.

Yuan Yao, Jiaju Du, Yankai Lin, Peng Li, Zhiyuan Liu, Jie Zhou, and Maosong Sun. 2021. CodRED: A cross-document relation extraction dataset for acquiring knowledge in the wild. In *EMNLP*.

Yuan Yao, Deming Ye, Peng Li, Xu Han, Yankai Lin, Zhenghao Liu, Zhiyuan Liu, Lixin Huang, Jie Zhou, and Maosong Sun. 2019. DocRED: A large-scale document-level relation extraction dataset. In *ACL*.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *ICML*.

Bowen Zhang and Harold Soh. 2024. Extract, define, canonicalize: An llm-based framework for knowledge graph construction. In *EMNLP*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics.

Yuqi Zhu, Xiaohan Wang, Jing Chen, Shuofei Qiao, Yixin Ou, Yunzhi Yao, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Llms for knowledge graph construction and reasoning: recent capabilities and future opportunities. *World Wide Web*, 27.

A Datasets and Ontology

Question answering datasets. We use the same dataset splits of HotpotQA (Yang et al., 2018) and MuSiQue (Trivedi et al., 2022) used in Gutiérrez et al. (2025); Chepurova et al. (2026). Each split is a subset of the original dataset consisting of 1000 entries each. Each dataset entry contains a list of text paragraphs as well as a question expressed in natural language. All KG extraction methods are run on each text paragraph, and the multiple KGs obtained for each entry are merged by also leveraging the respective entity deduplication steps. Finally, before running our QA experiments, the KGs extracted from each entry are merged.

Ontology. We extract a subset of entity types, predicates, and ontology constraints from Wikidata (Vrandečić and Krötzsch, 2014). We do so by re-adapting the code by Chepurova et al. (2026) using SPARQL queries run on the public Wikidata endpoint. That is, we also extract constraints defined over qualifiers (see Section 2). That is, for each predicate r, we also query for the set of allowed qualifier predicates, if any. In Table A.1 we show statistics about the retrieved ontology, such as the number of types, predicates, and constraints. If a predicate does not specify constraints, we assume it has none. Finally, we ensure we use the same ontology fragment for the baselines WIKONTIC and IN-CONTEXT considered in Section 5.

Ontology Elements			Ontology Constraints for Predicates		
$\|\mathcal{T}\|$	$\|\mathcal{R}\|$	$\|\mathcal{Q}\|$	w/ Domain	w/ Range	w/ Allowed Qualifiers
3768	2700	786	1757	1926	496

Table A.1: **Statistics of the Wikidata ontology fragment used.** We show the number of entity types $|\mathcal{T}|$, the total number of predicates $|\mathcal{R}|$, and the number of qualifier predicates $\mathcal{Q} \subset \mathcal{R}$, i.e., $|\mathcal{Q}|$. In addition, we show how many predicates in $\mathcal{R}$ specify domain, range, and allowed qualifier predicates ontology constraints.

B Basic Graph Patterns Benchmark

In order to evaluate the quality of the KGs extracted by ontology-grounded methods, we compute the occurrence of *basic graph patterns* (BGPs) useful to build SPARQL queries. For example, a BGP consisting of two abstract triples and using predicates `occupation:` and `awardReceived:` is:

```
{ ?person occupation: ?occupation .
  ?person awardReceived: ?award . }
```

This abstract BGP corresponds to a sub-graph pattern (or graph motif) with three entity variables (`?person`, `?occupation`, `?award`) and two predicates (`occupation:` and `awardReceived:`), and can be the skeleton of a `WHERE` statement in a SPARQL query. Alternatively, following the literature of query answering on incomplete KGs (Gregucci et al., 2025), the same BGP can be rewritten as the existentially-qualified first-order formula:

$$\exists P, O, A\colon \mathsf{occupation}(P, O) \wedge \mathsf{awardReceived}(P, A).$$

The idea of our benchmark based on BGPs is to measure how many times this graph pattern matches with an extracted KG, i.e., counting the number of distinct assignments to the person, occupation and award entity variables. The higher is the number of matching BGPs, the higher is the likelihood that a SPARQL query having such BGP will return a non-empty result. In other words, the existence of BGPs in a KG is a necessary—although not sufficient—condition for retrieving relevant information by means of a SPARQL query.

This setting is similar to another one used to evaluate graph generative models, e.g., where the percentage of graphs having certain graph motifs as in molecules generation is computed (Li et al., 2018). Similarly, other works compare the divergence between the distribution of the number of graph motifs in the ground truth and in the generated graphs (Hocevar and Demsar, 2014; You et al., 2018). However, the difference is that in our case we do not have access to the ground truth KG.

Therefore, inspired by h-index and ik-index citation metrics rewarding an author for having many citations over a diverse set of papers, we use the same metrics to reward a KG extraction method for having a high number of matches over a diverse set of BGPs. In our case, a method achieves an h-index of n if there are at least n BGPs appearing with a frequency of n or above in the extracted KGs. Moreover, a method achieves an ik-index of n if there are at least n BGPs with a frequency equal or greater than k. To compute the number of matching BGPs, we filter out the extracted triples violating the ontology, and recover the frequency of each BGP using a SPARQL run on a qlever SPARQL endpoint. Next, we detail two datasets of BGPs we use: one extracted from real-world queries run on the Wikidata SPARQL endpoint, and one artificially constructed from the consid-



Figure A.1: **Examples of basic graph patterns (BGPs) structures associated to different queries** (the image is taken from Gregucci et al. (2025)). Each sub-graph structure corresponds to a particular type of multi-hop query. For instance, a query asking for "Which actor performed in a movie filmed in New York City" is a 2-hop SPARQL query (or 2p) having the following BGP with ground entities replaced with variables: { $?x1$ cast member: $?x2$. $?x2$ filming location: $?x3$. }, where cast member: (P161) and filming location: (P915) are two Wikidata predicates (Vrandečić and Krötzsch, 2014). We artificially construct BGPs having structures such as the ones above. As detailed in Appendix B, we do so by extracting such patterns from a special KG linking entity types in the ontology by means of domain-range constraints. We also consider alternative BGPs structures where the directionality of predicates is inverted (e.g., yielding the r-2i structure from 2i). Table B.2 shows a formalization of each BGP structure in SPARQL, as well as their number.

BGPs Dataset		**Artificial**	**LSQ-2.0**
# patterns		264259	5757
# predicates		266	716
# Triples	Min.	2	2
	Avg.	2.946	2.905
	Max.	3	14
Pattern degree (or # Joins)	Min.	2	2
	Avg.	2.000	2.724
	Max.	2	14

Table B.1: **Statistics of basic graph patterns (BGPs) from our artificially-constructed dataset and from the real-world Wikidata queries from LSQ-2.0.** We show the number of BGPs and the number of unique predicates. Moreover, we show the minimum/average/maximum number of triples per BPG, as well as the number of joins (i.e., the node degree of the BGP itself). The BGPs obtained from the LSQ-2.0 collection (Saleem et al., 2015; Stadler et al., 2024) consists of a more diverse set of predicates when compared to the artificially extracted BPGs (see Appendix B).

Kind	**Basic Graph Pattern Formalization**	**Count**
2p	?x1 ?r1 ?x2 . ?x2 ?r2 ?x3 .	6'427
2i	?x1 ?r1 ?x3 . ?x2 ?r2 ?x3 .	3'041
1p2i	?x1 ?r1 ?x2 . ?x2 ?r2 ?x4 . ?x3 ?r3 ?x4 .	10'000
2i1p	?x1 ?r1 ?x3 . ?x2 ?r2 ?x3 . ?x3 ?r3 ?x4 .	10'000
3p	?x1 ?r1 ?x2 . ?x2 ?r2 ?x3 . ?x3 ?r3 ?x4 .	10'000
3i	?x1 ?r1 ?x4 . ?x2 ?r2 ?x4 . ?x3 ?r3 ?x4 .	10'000
r-2i	?x1 ?r1 ?x2 . ?x1 ?r2 ?x3 .	4'791
r-1p2i	?x1 ?r1 ?x2 . ?x1 ?r2 ?x3 . ?x2 ?r3 ?x4 .	10'000
r-2i1p	?x1 ?r1 ?x2 . ?x2 ?r2 ?x3 . ?x2 ?r3 ?x4 .	10'000
r-3i	?x1 ?r1 ?x2 . ?x1 ?r2 ?x3 . ?x1 ?r3 ?x4 .	10'000

Table B.2: **Artificial BGPs structures and their SPARQL formalization.** We extract a number of BGPs for each structure starting from an ad-hoc graph consisting of entity types as nodes where links encodew domain-range constraints (see Appendix B). For computational reasons, we limit the number of BGPs to 10^4.

ered domain-range ontology constraints. Table B.1 shows statistics about both BGPs datasets.

Real-world BGPs. We extract BGPs from real-world SPARQL queries on Wikidata, which are collected under the Linked Dataset SPARQL Query Logs (LSQ-2.0) (Saleem et al., 2015; Stadler et al., 2024). We do so by running SPARQL queries on the LSQ-2.0 publicly available endpoint. After we collect a BGP, we replace constant entities in it with fresh variables and filter it out only if it contains multiple triples with the same predicate. This allows us to only keep interesting BGPs that do not map to a specific SPARQL query with bound entities. We then remove duplicate BGPs via a directed graph isometry test. The obtained dataset consists of 5757 BGPs over 716 Wikidata predicates.

Artificial BGPs. To build a larger dataset of BGPs, we artificially construct them by exploiting common Wikidata predicates and BGP structures commonly used in existing complex query answering benchmarks (Gregucci et al., 2025). We start by building a special KG—called *ontology KG*—encoding the domain-range constraints over a set of popular Wikidata predicates, which we take from the DocRED (Yao et al., 2019) and CodRED (Yao et al., 2021) datasets. That is, we construct a KG such that for each predicate r specifying at least one between the domain and range constraints, we insert a triple (t_s, r, t_o) where $t_s \in \mathsf{dom}(r)$ (resp. $t_o \in \mathsf{rng}(r)$). If the domain (resp. range) is *not* specified, we introduce a fresh *"any"* type node in the ontology KG. Then, we run SPARQL queries on such KG in order to exhaustively recover BGPs with ground predicates having various structures corresponding to complex queries. Fig. A.1 shows the different BGP structures we consider useful to construct SPARQL queries. For computational reasons we randomly sample up to $10'000$ BGPs per structure. This yields a set of BGPs having various kinds of structures and with different com-

binations of predicates that are consistent withe domain-range ontology constraints. We summarize the BGPs structures and their count in Table B.2.

C Detailed Description of the Qualifiers Correction Step

After we corrected the triples violating domain-range constraints in Section 4.1, we can now correct any attached qualifier violating the ontology. Similarly to our description of triples correction in Section 4.1, here we describe a collection of possible KG extraction mistakes that make qualifiers violating the ontology. Each of these possible mistakes will then motivate the corrective actions that an LLM will be prompted to choose from.

V4. A qualifier (r_q, o_q) that is attached to a triple (s, r, o) is inconsistent because r_q is **not a qualifier predicate allowed by the triple predicate** r, i.e., $r_q \notin \mathsf{qual}(r)$. For example, in Wikidata we have that $\mathsf{location} \notin \mathsf{qual}(\mathsf{publication\ date})$, i.e., location is not a qualifier predicate allowed by publication date. Therefore, the qualifier $(\mathsf{location}, \mathsf{London})$ cannot be a consistent when attached to the triple $(\mathsf{Animal\ Farm}, \mathsf{publication\ date}, 1945)$. Instead, $\mathsf{place\ of\ publication} \in \mathsf{qual}(\mathsf{publication\ date})$ is an allowed qualifier and therefore it should be used instead of location as qualifier predicate. The corrective action here is *replacing the qualifier predicate with one allowed by the predicate in the triple*.

V5. A qualifier (r_q, o_q) of a triple (s, r, o) might not be consistent with respect to the range constraint of r_q, and the reason is that r_q is **a qualifier predicate that has been mis-canonicalized**. That is, we have that $r_q \in \mathsf{qual}(r)$, yet the types of the entity o_q are incosnsistent w.r.t. the range of r. For example, consider the qualifier $(\mathsf{distribution\ format}, \mathsf{Penguin\ Books})$ attached $(\mathsf{Animal\ Farm}, \mathsf{publication\ date}, 1945)$. This qualifier is inconsistent if $\tau(\mathsf{Penguin\ Books}) = \{\mathsf{publisher}\}$, since it would not satisfy the range constraint of distribution format, i.e., expecting a format. Yet, we have that $\mathsf{distribution\ format} \in \mathsf{qual}(\mathsf{publication\ date})$. The correcting action is replacing the qualifier predicate with publisher, which is allowed by the predicate publication date and it is consistent with the type of Penguin Books.

V6. Finally, similarly to the underspecified or missing entity types mistake discussed for triples in Section 4.1, a qualifier (r_q, o_q) might not be consistent because of the same reason, i.e., the types of o_q are not specific enough, or a type is missing, in order to be consistent with the range of r_q.

As detailed in Section 4.2, these possible mistakes suggest us to prompt an LLM to select between two syntactic transformations of the qualifier, namely replace_predicate and add_object_type.

D LLM-based Baseline with In-Context Ontology Constraints

We detail our KG extraction baseline inserting the ontology constraints as part of the input context, which we denote simply as IN-CONTEXT, and report the prompts in Appendix G.3.

The idea of IN-CONTEXT is to (1) firstly predict a relevant fragment of the ontology consisting of predicates and allowed qualifiers given the source text; and (2) extract the KG from the source text by instructing the LLM to comply with the ontology. For (1), we prompt an LLM to extract open-domain predicates given the source text. Then, the extracted open-domain predicates are mapped to the top-k most similar predicates allowed by ontology, in terms of embedding similarity computed over predicate labels and aliases. We use $k = 20$ in our experiments. Then, for (2) we insert a verbalization of the predicted predicates, relative domain-range constraints, and allowed qualifiers as part of the LLM context upon KG extraction. If an extracted predicate (resp. entity type) is not one of the allowed ones (resp. one appearing in a domain or range of a predicate), we still keep it as an *unknown* predicate (resp. entity type) labeled with the same name.

E Human Qualitative Evaluation

We perform a qualitative analysis using two of the authors of this paper to understand whether OAK+MEND fixes the ontology violations by sacrificing the semantic faithfulness of triples w.r.t. the source text. To do so, we focus on HotpotQA and give each human evaluator 100 triples extracted and corrected using QWEN3-30B-A3B, and 100 triples extracted and corrected using GPT-OSS-120B. The qualitative evaluation is therefore performed by considering a total of 400 triples that have been randomly sampled from the KGs.

Evaluation format. A human evaluator receives the following information about a triple correction:

- The triple violating the ontology constraints, i.e., extracted by OAK and *before* the correction with MEND.

- The types of subject and object entities *before* the correction.
- The triple satisfying the ontology constraints, i.e., *after* the correction with MEND.
- The types of subject and object entities *after* the correction.
- The source text fragment where the triple was originally extracted from.

Then, we ask the evaluator to answer either *Yes* or *No* to the following questions while considering the given source text fragment:

- Is the inconsistent triple *before* the correction faithful?
- Are the subject and object entity types *before* the correction faithful?
- Is the consistent triple *after* the correction faithful?
- Are the subject and object entity types *after* the correction faithful?

In other words, each triple correction is checked against the source text for semantic faithfulness by one annotator using binary metrics.

Results. We then compute the percentage of triples and the related entity types that are regarded as faithful *before* and *after* the predicted correction. Although brief, our qualitative results reported in Table E.1 suggest that the rate of text-based faithfulness of triples increases after the corrections. Meanwhile, the faithfulness of the subject and object entity types remains approximately the same. These results focusing on semantic faithfulness complement our experiments on downstream QA (see Section 5.2), where we quantitatively show that applying post-extraction corrections comes with little to no performances loss.

	QWEN3-30B-A3B	GPT-OSS-120B
Before the correction		
% faithful triples	37.0	40.5
% triples w/ faithful types	86.0	79.0
After the correction		
% faithful triples	65.5	67.0
% triples w/ faithful types	85.5	80.0

Table E.1: **Our framework based on post-extraction corrections of ontology violations does not sacrifice semantic faithfulness.** On HotpotQA, we show the percentage of triples and subject and object entity types that are regarded as faithful w.r.t. the source text in our human audit *before* and *after* the corrections (see Section 4.1). On both LLMs that we considered in our experiments, the percentage of faithful triples increases after the corrections, while the percentage of triples whose both subject and object entity types are text-aligned remains approximately the same.

F Additional Experimental Results

Table F.1 shows KG extraction statistics, ontology consistency metrics and token cost achieved on MuSiQue instead. The observations made for HotpotQA in Section 5.1, i.e., competitive ontology consistency rates with a modest token cost, translate to MuSiQue as well. Table F.2 reports QA results ablated by the qualifiers presence, confirming their importance in capturing crucial semantic information from the text. In Tables F.3 and F.4 we separate the results obtained on LSQ-2.0-based and artificial BGP datasets (see Appendix B). Finally, in Table F.6 we provide a qualtiative analysis of triples extracted by the IN-CONTEXT baseline on QWEN3-30B-A3B, confirming our claim in Section 5.3 that triples containing redundant information are extracted, thus boosting h-index metrics on our BGPs benchmark.

Hardware, time, LLMs and repetitions. All experiments are executed on NVIDIA RTX A6000 and A100 GPUs, and results can be reproduced in a about a week. We use the default sampling hyperparameters for all LLMs. We do so by using vLLM to serve our models. All the experimental results we show in this paper are obtained by averaging across 3 independent runs.

Dataset: MuSiQue	Triples Statistics		Qualifiers Statistics		# Tokens ($\times 10^6$)		
	# Total	% Valid	# Total	% Valid	Prompt	Completion	Weighted
QWEN3-30B-A3B							
KGGEN	283 729 ± 992	—	0	—	140.6	20.7	55.8 (0.66)
WIKONTIC	204 689 ± 207	72.9 ± 0.2	88 209 ± 540	—	260.8	18.9	84.1 (1.00)
IN-CONTEXT	345 328 ± 1621	78.2 ± 0.2	75 205 ± 505	83.3 ± 0.1	176.4	25.5	69.6 (0.83)
OAK	224 294 ± 619	58.7 ± 0.2	89 503 ± 1563	29.3 ± 0.1	65.5	14.0	**30.4** (0.36)
OAK+MEND	222 749 ± 636	**95.7 ± 0.1**	89 503 ± 1563	**90.1 ± 0.2**	80.0	12.5	32.5 (0.39)
GPT-OSS-120B							
KGGEN	230 359 ± 6556	—	0	—	166.3	103.7	145.3 (0.83)
WIKONTIC	189 396 ± 517	**99.1 ± 0.1**	79 261 ± 375	—	269.6	101.8	176.0 (1.00)
IN-CONTEXT	112 207 ± 713	97.3 ± 0.1	18 610 ± 407	**98.0 ± 0.1**	181.9	39.5	85.0 (0.48)
OAK	195 568 ± 787	63.3 ± 0.1	85 701 ± 484	32.2 ± 0.3	66.6	54.8	**71.4** (0.41)
OAK+MEND	194 539 ± 800	97.3 ± 1.0	85 701 ± 484	94.8 ± 0.1	135.4	102.6	136.5 (0.78)

Table F.1: **OAK+MEND achieves high ontology consistency rates for both triples and qualifiers, yet requiring modest token cost.** We show the number of triples and qualifiers extracted by different KG construction methods, as well as the percentage consistent with ontology constraints (see Section 2), where applicable. We also show the total number of prompt and completion tokens (in millions), and a weighted cost combination that takes into account a prompt-to-completion cost ratio of $1/4$ (e.g., as the $ price ratio for GPT-4.1).

Table F.2: **The absence of qualifiers significantly reduces QA performances.** We show the drop of exact match and F_1 metrics for QA achieved by different KG extraction methods and on different datasets, where we remove the triples qualifiers.

Method	HotpotQA		MuSiQue	
	GPT-OSS-120B			
	EM	F_1	EM	F_1
IN-CONTEXT	33.1 ± 0.7	41.8 ± 0.8	17.5 ± 0.5	23.7 ± 0.7
– QUALIFIERS	32.8 ± 0.5	40.2 ± 0.4	16.1 ± 1.0	21.9 ± 1.3
WIKONTIC	40.4 ± 0.8	50.4 ± 0.8	19.9 ± 0.2	28.0 ± 0.6
– QUALIFIERS	35.8 ± 0.3	44.9 ± 0.1	17.2 ± 0.5	23.3 ± 0.5
OAK+MEND	**46.1 ± 2.0**	**56.8 ± 1.7**	**21.5 ± 0.1**	**29.9 ± 0.2**
– QUALIFIERS	38.5 ± 1.2	48.1 ± 1.4	17.4 ± 0.3	23.4 ± 0.1

Method	LSQ-2.0 Patterns			Artificial Patterns		
	h-index	i10-index	i100-index	h-index	i10-index	i100-index
QWEN3-30B-A3B						
WIKONTIC	319.7 ± 0.6	1653.3 ± 17.2	699.3 ± 10.1	132.3 ± 1.2	1542.7 ± 40.4	196.0 ± 9.6
IN-CONTEXT	**469.7 ± 3.2**	**2426.7 ± 35.5**	**1186.7 ± 3.8**	**237.7 ± 5.9**	**3492.3 ± 105.3**	**634.3 ± 22.9**
OAK	315.0 ± 1.0	1626.3 ± 17.5	707.7 ± 1.2	129.0 ± 1.7	1299.7 ± 19.2	177.0 ± 4.0
OAK+MEND	377.0 ± 1.7	1957.3 ± 20.1	866.7 ± 11.7	158.0 ± 0.1	2204.7 ± 20.0	275.0 ± 8.7
GPT-OSS-120B						
WIKONTIC	346.3 ± 3.5	**1884.0 ± 20.1**	**822.3 ± 9.5**	**148.7 ± 2.5**	**1921.0 ± 36.5**	**245.7 ± 10.1**
IN-CONTEXT	293.7 ± 1.5	1492.7 ± 11.0	607.7 ± 2.5	127.7 ± 2.5	1051.7 ± 18.0	167.7 ± 7.6
OAK	306.3 ± 2.5	1467.0 ± 14.0	653.3 ± 18.0	130.3 ± 1.2	1152.0 ± 41.7	169.7 ± 2.1
OAK+MEND	**359.0 ± 6.1**	1835.0 ± 44.8	**813.0 ± 22.0**	**150.3 ± 3.2**	1863.7 ± 17.6	**241.7 ± 3.5**

Table F.3: **Breakdown of the basic graph pattern (BGPs) matching results on the HotpotQA data set.** We show the total number of pattern matches (N), and the h-index, i10-index, and i100-index scores (higher is better).

Method	LSQ-2.0 Patterns			Artificial Patterns		
	h-index	i10-index	i100-index	h-index	i10-index	i100-index
QWEN3-30B-A3B						
WIKONTIC	325.0 ± 3.0	1778.0 ± 9.5	762.7 ± 2.9	140.0 ± 2.0	1606.7 ± 21.1	214.3 ± 4.0
IN-CONTEXT	**582.0 ± 2.6**	**2797.0 ± 13.2**	**1519.0 ± 14.5**	**296.7 ± 9.0**	**4829.7 ± 64.6**	**972.7 ± 22.2**
OAK	344.3 ± 0.6	1779.7 ± 19.1	797.7 ± 5.7	146.0 ± 1.0	1459.7 ± 25.5	234.0 ± 6.1
OAK+MEND	404.3 ± 1.5	2153.3 ± 14.2	994.7 ± 2.1	171.0 ± 1.0	2640.7 ± 20.8	352.3 ± 2.5
GPT-OSS-120B						
WIKONTIC	**384.0 ± 3.6**	**2076.7 ± 29.4**	**913.7 ± 4.2**	**152.7 ± 4.2**	**2205.7 ± 45.8**	**268.3 ± 1.5**
IN-CONTEXT	314.7 ± 5.5	1689.0 ± 8.5	713.0 ± 3.0	134.7 ± 3.1	1186.3 ± 18.9	189.0 ± 5.2
OAK	324.3 ± 1.2	1620.7 ± 5.0	711.7 ± 1.5	135.3 ± 0.6	1249.3 ± 3.2	190.0 ± 4.4
OAK+MEND	**385.7 ± 1.5**	**2078.0 ± 18.5**	899.7 ± 9.1	**153.3 ± 2.1**	2057.0 ± 42.4	257.3 ± 4.9

Table F.4: **Breakdown of the basic graph pattern (BGPs) matching results on the MuSiQue data set.** We show the total number of pattern matches (N), and the h-index, i10-index, and i100-index scores (higher is better).

Method	HotpotQA		MuSiQue	
	i1-index	i5-index	i1-index	i5-index
QWEN3-30B-A3B				
WIKONTIC	8383.7 ± 84.0	4436.0 ± 58.8	8246.0 ± 29.5	4603.0 ± 36.5
IN-CONTEXT	**13 353.3 ± 120.7**	**7863.3 ± 133.5**	**15 639.7 ± 125.9**	**9837.7 ± 34.5**
OAK	7178.0 ± 37.6	3997.7 ± 52.2	7270.7 ± 43.4	4248.7 ± 9.7
OAK+MEND	10 378.3 ± 54.2	5744.3 ± 43.4	11 146.7 ± 77.7	6434.0 ± 39.5
GPT-OSS-120B				
WIKONTIC	**10 037.3 ± 179.5**	**5315.7 ± 62.1**	**11 172.3 ± 222.6**	**5957.0 ± 76.9**
IN-CONTEXT	6001.7 ± 95.3	3385.7 ± 17.9	6416.7 ± 46.7	3803.3 ± 56.1
OAK	6587.7 ± 137.8	3573.0 ± 30.8	6728.3 ± 68.3	3783.0 ± 31.2
OAK+MEND	**9814.0 ± 140.6**	5170.3 ± 42.1	10 383.7 ± 115.8	5633.0 ± 35.5

Table F.5: **Basic graph patterns (BGPs) matching results focusing more on graph patterns diversity.** We complement the results showed in Table 4 with i1-index and i5-index, which reward a wide BGP coverage more when compared to BGP matchings frequency as measured with h-index and i100-index metrics. For instance, i1-index is the number of BGPs in both LSQ-2.0 and artificial patterns datasets (see Appendix B) that have at least one occurrence in the KGs extracted by the different methods. Therefore, the higher the i1-index is, the better is the coverage of different BGPs by a KG construction method.

Common Jointly--extracted Predicates	Count	Example Source Text	Example of Extracted Triples
location of creation (P1071) country of origin (P495)	598	The Magic Christmas Tree is a 1964 American Christmas-themed fantasy-adventure film about a boy who uses a magic ring to bring a Christmas tree to life.	(The Magic Christmas Tree, P1071, U.S.) (The Magic Christmas Tree, P495, U.S.)
Explanation. The predicate location of creation (P1071) is used to specify the physical location of production of an item, while country of origin (P495) specifies the attributed nationality. While the country of origin is specified by the source text (i.e., by the word "American") the location of creation is not, yet triples with both predicates are extracted by the LLM.			
commercialization date (P5204) publication date (P577)	242	Beautiful Eyes is the second extended play (EP) by American singer-songwriter Taylor Swift. The EP was released on July 15, 2008 by Big Machine Records exclusively to Walmart stores in the United States and online.	(Beautiful Eyes, P5204, 15 July 2008) (Beautiful Eyes, P577, 15 July 2008)
Explanation. The predicate commercialization date (P5204) is typically used to specify the date on which a certain product or invention is firstly commercialized. Instead, publication date (P577) specifies the date on which a creative work such as a musical album is made public. While P5204 and P577 are substantially different predicates and only one of them should be selected, the LLM extracts two triples with both predicates yet encoding the same fact.			
location of creation (P1071) country of origin (P495) location of formation (P740)	21	Liberty Tax Service is an American company specializing in the preparation of tax returns for individuals and small businesses. [...] The company began in Canada in 1997 when John Hewitt, co-founder of Jackson Hewitt, acquired a Canadian tax franchisor, U&R Tax Depot.	(Liberty Tax, Inc., P1071, Canada) (Liberty Tax, Inc., P495, Canada) (Liberty Tax, Inc., P740, Canada)
Explanation. The predicate location of creation (P1071) is used to specify the physical location of production of an item, while country of origin (P495) specifies the attributed nationality. Instead, location of formation (P740) specifies where an organization was originally established. Instead of selecting only a single predicate expressing the fact that Liberty Tax, Inc. started in Canada (i.e., P740), the LLM selected all predicates and extracted multiple triples encoding the same fact.			

Table F.6: **The IN-CONTEXT baseline with QWEN3-30B-A3B erroneously extracts multiple triples with different predicates to encode a single fact, hence artificially inflating performances on SPARQL patterns** (see Section 5.3). We show three sets of predicates that are often used together in multiple triples encoding the same information from the source text. The redundancy of the extracted triples increases the average edge multiplicity, and therefore it artificially inflates the frequencies of basic graph patterns (BGPs) (as shown in Table 4).

G Prompts

G.1 OAK

Open-domain KG extraction

```
## Task Description
Your task is to extract a Wikidata-like
knowledge graph from text. A knowledge graph
consists of subject-predicate-object
triples where:
- subject and object: Named entities or
concepts that describe groups of people,
events, or any abstract objects.
- predicate: A predicate (or relation type)
that connects the subject and object.

Additionally, some triples may have
qualifiers providing contextual information
about a triple (e.g., date, place, or other
attributes).
Each qualifier only exists when linked to a
triple and consists of a predicate-object
pair.

## Inputs
You will receive the text.

## Output Format
Extract triples and qualifiers in **JSON**
as a list of dictionaries, where each
dictionary contains:
- "triple": A list of three elements:
subject, predicate, and object
- "subject_type": The type of the subject
- "object_type": The type of the object
- "qualifiers": An optional list of
dictionaries, where each dictionary
contains:
    - "pair": A list of two elements:
    qualifier predicate, and qualifier
    object
    - "object_type": The type of the
    qualifier object

## Example
<... an example here ...>
```

Entity Type Canonicalization

```
## Task Description
Your task is to disambiguate an entity type
for one or more entities extracted from
text.

## Inputs
You will receive a text, a list of extracted
entities, an extracted entity type, and a
list of candidate types.
Return a single type for all the entities.
No additional text.
```

Predicate Canonicalization

```
## Task Description
Your task is to disambiguate a predicate
indicating the relationship type between
pairs of entities extracted from text.
Assume the predicate direction is not
important.

## Inputs
You will receive a text, a list of extracted
entity pairs, the extracted predicate, and a
list of candidate predicates.
Return a single predicate for all the entity
pairs. No additional text.
```

Entity Deduplication

```
## Task Description
Find duplicates for a given entity, and an
entity alias that best represents the
duplicates.
Duplicates are those that are the same in
meaning, such as with variation in tense,
plural form, stem form, case, abbreviation,
shorthand.

## Inputs
You will receive an entity and a list of
candidate duplicate entities.

## Output Format
Return duplicates and the alias in **JSON**
as a dictionary containing:
- "duplicates": A list of duplicate entities
taken from the candidates list
- "alias": Best entity name to represent the
duplicates

If there are no duplicates, then return the
empty dictionary {}.
```

G.2 OAK+MEND

Triple Correction

```
## Task Description
You are a knowledge graph expert. Your task
is to correct an invalid
subject-predicate-object (SPO) triple so
that it satisfies ontology constraints,
while still reflecting information present
in the source text.

## Inputs
- Source text
- The invalid triple (subject, predicate,
object)
- The domain and range constraints of the
predicate
- A list of candidate replacement predicates
- The reasons why the triple is invalid

## Allowed Actions (applied in sequence)
- "swap": swap subject and object
- "add_subject_type": add type to subject
(to satisfy domain)
- "add_object_type": add type to object (to
satisfy range)
```

```
- "replace_predicate": replace predicate
with a candidate

## Rules
- Added types must be plausible given the
text
- Only use candidates for replace_predicate
- Actions are applied in order, each
modifying the result of the previous one
- Choose the sequence of actions that
produces a triple consistent with the
ontology constraints and grounded in the
source text. If no such sequence exists,
return an empty list []

## Output
Return only a **JSON** list of [action,
value] pairs, where action is one of swap,
add_subject_type, add_object_type,
replace_predicate, and value is the new type
or predicate string (or null for swap).
Return an empty list if no correction is
possible.

### Output Examples
- [["replace_predicate", "publication
date"]]
- [["swap", null], ["add_object_type",
"person"]]
- []
```

Qualifier Correction

```
## Task Description
You are a knowledge graph expert. Your task
is to correct a qualifier predicate-object
pair associated to a triple so that it
satisfies ontology constraints, while still
reflecting information present in the source
text.

## Inputs
- Source text
- The triple (subject, predicate, object)
- The invalid qualifier (qualifier
predicate, qualifier object)
- The range constraint of the qualifier
predicate
- A list of candidate replacement qualifier
predicates
- The reason why the qualifier is invalid

## Allowed Actions
- "add_object_type": add type to qualifier
object (to satisfy range)
- "replace_predicate": replace qualifier
predicate with a candidate

## Rules
- Added types must be plausible given the
text
- Only use candidates for replace_predicate
- Choose the action that produces a
qualifier consistent with the ontology
constraints and grounded in the source text.
If no action exists, return an empty list []

## Output
Return only a **JSON** list [action, value],
where action is one of add_object_type,
replace_predicate, and value is the new type
or predicate string. Return an empty list if
no correction is possible.

### Output Examples
- ["replace_predicate", "doctoral advisor"]
- ["add_object_type", "person"]
- []
```

G.3 IN-CONTEXT Baseline

Extract Predicates

```
## Task Description
Extract Wikidata-like predicates (or
relationship types) explaining the
relationships between the entities in the
given text.

## Inputs
You will receive the text.

## Output Format
Return the predicates in **JSON** as a list
of strings.

## Example
Text:
Alan Turing (23 June 1912 – 7 June 1954) was
an English mathematician. He received the
Smith's Prize in 1936, and in 1938 he earned
a doctorate degree from Princeton
University, advised by Alonzo Church.

### Output
[
    "date of birth",
    "date of death",
    "country of citizenship",
    "occupation",
    "award received",
    "educated at"
]
```

Extract the Knowledge Graph

```
## Task Description
Your task is to extract a Wikidata-like
knowledge graph from text. A knowledge graph
consists of subject-predicate-object
triples where:
- subject and object: Named entities or
concepts that describe groups of people,
events, or any abstract objects.
- predicate: A predicate (or relation type)
that connects the subject and object.

Additionally, some triples may have
qualifiers providing contextual information
about a triple (e.g., date, place, or other
attributes).
Each qualifier only exists when linked to a
triple and consists of a predicate-object
pair.
```

```
### Task Constraints
You are allowed to use only the provided
predicates.
Most importantly, each provided predicate
comes with its own set of allowed domain and
range entity types.
For each triple, the subject type must be
one (or a specialization) of any type in the
domain list specified by the predicate.
In addition, the object type must be one (or
a specialization) of any type in the range
list specified by the predicate.
If no types are specified as domain or range,
then there are no type constraints for the
subject or object entity, respectively.

Furthermore, you are allowed to use only the
provided qualifiers.
Most importantly, each predicate comes with
its own list of allowed qualifier
predicates.
In addition, the qualifier object type must
be one (or a specialization) of any type in
the range list specified by the qualifier
predicate.
If no types are specified as range, then
there are no type constraints for the
qualifier object entity.

## Inputs
You will receive the text, the list of
allowed predicates each with their own
domain-range and permitted qualifiers
constraints, and the list of allowed
qualifiers each with their own range
constraint.

## Output Format
Extract triples and qualifiers in **JSON**
as a list of dictionaries, where each
dictionary contains:
- "triple": A list of three elements:
subject, predicate, and object
- "subject_type": The type of the subject
- "object_type": The type of the object
- "qualifiers": An optional list of
dictionaries, where each dictionary
contains:
    - "pair": A list of two elements:
    qualifier predicate, and qualifier
    object
    - "object_type": The type of the
    qualifier object

## Example
<... an example here ...>
```