LLMs4OL 2025: The 2nd Large Language Models for Ontology Learning Challenge at the 24th ISWC

LLMs4OL 2025 Task Participant Long Papers

https://doi.org/10.52825/ocp.v6i.2887

© Authors. This work is licensed under a Creative Commons Attribution 4.0 International License

Published: 01 Oct. 2025

SBU-NLP at LLMs4OL 2025 Tasks A, B, and C: Stage-Wise Ontology Construction Through LLMs Without any Training Procedure

Rashin Rahnamoun[1,*] and Mehrnoush Shamsfard[1]

[1]Shahid Beheshti University, Tehran, Iran

*Correspondence: Rashin Rahnamoun, rahnamounrashin@gmail.com

Abstract. Automated ontology construction is a challenging task that traditionally requires extensive domain expertise, data preprocessing, and resource-intensive model training. While learning-based methods with fine-tuning are common, they often suffer from high computational costs and limited generalizability across domains. This paper explores a fully automated approach that leverages powerful large language models (LLMs) through prompt engineering, eliminating the need for training or fine-tuning. We participated in the LLMs4OL 2025 shared task, which includes four subtasks: extracting ontological terms and types (Text2Onto), assigning generalized types to terms (Term Typing), discovering taxonomic relations (Taxonomy Discovery), and extracting non-taxonomic semantic relations (Non-Taxonomic Relation Extraction). Our team focused on the first three tasks by using stratified random sampling, simple random sampling, and chunking-based strategies to include training sets in the prompts without limitations imposed by context window sizes. This simple yet general approach has proven effective across these tasks, enabling high-quality ontology construction without additional annotations or training. Additionally, we show that pretrained sentence embedding models ranging from 0.1B to 1.5B parameters perform comparably to a simple F1 token overlap baseline in taxonomy discovery, suggesting that embedding-based methods may not always offer significant advantages. Our findings highlight that prompt-based strategies with modern LLMs enable efficient, scalable, and domain-independent ontology construction, providing a promising alternative to traditional, resource-heavy methods.

Keywords: Automated Ontology Construction, Large Language Models, Prompt Engineering

1. Introduction

Automated ontology construction is a complex and multifaceted task that traditionally involves extensive domain knowledge, data preprocessing, and model training. While learning-based methods—particularly those involving model fine-tuning—have become common, they often require significant computational resources and are not easily generalizable across domains. Ideally, a high-quality, fully automated approach that

bypasses model training and fine-tuning would offer a more efficient and accessible alternative.

The emergence of powerful large language models (LLMs) presents a promising opportunity to replace or supplement human experts in ontology creation. These models, with their ability to understand and generate structured knowledge from text, enable a new generation of ontology engineering tools that are fast, cost-effective, and domain-independent.

To explore the viability of LLM-based ontology construction, we participated in the LLMs4OL 2025 shared task, the second iteration of this competition following its debut in 2024 [1], [2], [3], [4]. The 2025 edition includes four distinct tasks: Task A – Text2Onto, which involves extracting ontological terms and their types from raw text; Task B – Term Typing, which focuses on assigning generalized types to lexical terms; Task C – Taxonomy Discovery, which requires identifying taxonomic (is-a) relations between types; and Task D – Non-Taxonomic Relation Extraction, which involves extracting other semantic relations between types.

Our team focused on Tasks A, B, and C, applying a unified prompting-based methodology that avoids any form of supervised training or fine-tuning. Instead, we leveraged prompt engineering strategies informed by sampling techniques from the provided training data. This approach allowed us to generate high-quality ontological structures using zero-shot or few-shot prompting, without incorporating any external knowledge or additional annotations.

Furthermore, in the Taxonomy Discovery subtask, we show that pretrained sentence embedding models (ranging from 0.1B to 1.5B parameters) perform comparably to a simple F1 token overlap baseline. This suggests that, in certain ontology tasks, embedding-based approaches may not provide a significant performance advantage over more lightweight lexical methods.

These findings highlight the practical benefits of prompt-based strategies in ontology learning tasks, especially when aiming to reduce reliance on computationally expensive training pipelines. Our results demonstrate that, with careful prompt engineering, modern LLMs can effectively perform complex ontology construction tasks in a scalable and resource-efficient manner. Achieved the best overall performance across all teams in the challenge and ranked 1st on the final leaderboard. This opens the door to more accessible ontology generation workflows, particularly in low-resource or time-constrained settings. Overall, our work contributes to the growing evidence that LLMs, when coupled with generalizable prompting strategies, can serve as a viable and efficient alternative to traditional learning-based ontology engineering approaches.

2. Datasets

The datasets presented in Table 1 correspond to those we actually used to produce the results reported in this paper. Note that these may not cover all datasets available within the respective tasks, but only those included in our experiments for Tasks A (Text2Onto), B (Term Typing), and C (Taxonomy Discovery).

Task A involves ontology extraction from domain-specific corpora in ecology, engineering, and scholarly domains.

Task B focuses on semantic term typing with datasets from OBI and SWEET taxonomies, along with two blind test sets (B5 and B6) evaluated without training data.

Task C targets taxonomy discovery using ontologies such as OBI, Schema.org, SWEET, and MatOnto. Here, test data counts represent unique types rather than document IDs.

Table 1. *Training and test data statistics for Tasks A, B, and C. Train/test numbers indicate unique document IDs, except for Task C test sets, where values refer to the number of unique types.*

Task & Dataset	Training Data	Testing Data
Task A - Text2Onto (Ecology)	2,000	483
Task A - Text2Onto (Engineering)	83	21
Task A - Text2Onto (Scholarly)	40	10
Task B - Term Typing (OBI)	201	87
Task B - Term Typing (MatOnto)	85	37
Task B - Term Typing (SWEET)	1,707	626
Task B - Term Typing (B5-Blind)	–	46
Task B - Term Typing (B6-Blind)	–	288
Task C - Taxonomy Discovery (OBI)	10,003	2,821
Task C - Taxonomy Discovery (SchemaOrg)	742	359
Task C - Taxonomy Discovery (SWEET)	11,626	4,118
Task C - Taxonomy Discovery (MatOnto)	885	370
Task C - Taxonomy Discovery (PO)	2005	916

The datasets used across the three tasks exhibit significant variability in size and complexity, which poses unique challenges for ontology learning methods. Task A's Text2Onto datasets differ notably in scale, with the ecology domain providing the largest corpus, potentially offering richer contextual information for ontology extraction compared to the relatively small engineering and scholarly datasets. Task B's Term Typing datasets include both well-established ontologies like OBI and SWEET with substantial training samples, alongside blind test sets (B5 and B6) that require robust generalization without prior training data. This setup tests the adaptability of models to unseen domains.

Task C's Taxonomy Discovery datasets are generally large, with tens of thousands of training instances reflecting comprehensive domain coverage. The distinction that test sets for Task C represent unique types rather than documents highlights the need for models to accurately infer hierarchical relationships at a fine-grained semantic level.

3. Methodology

3.1 Task A - Text2Onto

In this task, our goal is to extract terms and their corresponding types from given sets of documents. We employ two different prompt formats, each designed specifically for term extraction and type extraction, respectively, to effectively leverage LLMs. Also, two forms of prompting were used. Our approach leverages in-context learning by utilizing few-shot examples based on the training sets.

3.1.1 Term Extraction

Preprocessing. Let $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ denote the set of documents, where each document d_i is associated with an identifier id_i, title t_i, and text content x_i. Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_M\}$ be the set of extracted terms. We define a mapping function

$$f : \mathcal{T} \to 2^{\mathcal{D}} \tag{1}$$

which maps each term τ_j to a subset of documents $f(\tau_j) \subseteq \mathcal{D}$ in which the term appears. Using this, we construct the inverse mapping

$$g : \mathcal{D} \to 2^{\mathcal{T}} \tag{2}$$

such that for each document d_i, the associated term set is

$$g(d_i) = \{\tau_j \mid d_i \in f(\tau_j)\}. \tag{3}$$

From $\mathcal{D}$, a random subset $\mathcal{D}_s \subseteq \mathcal{D}$ of size $|\mathcal{D}_s| = k$ (where k is the sample size) is selected to form the prompt examples. Each sampled document $d_i \in \mathcal{D}_s$ is paired with its corresponding terms $g(d_i)$, which together constitute the few-shot exemplars for guiding the large language models. This preprocessing step ensures that the exemplars are representative and diverse, improving the efficacy of in-context learning for ontology construction.

Simple Random Sampling. In the preprocessing step, a random subset $\mathcal{D}_s \subseteq \mathcal{D}$ of size $|\mathcal{D}_s| = k$ is selected without replacement from the full document set $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$, where $N = |\mathcal{D}|$. The sampling is uniform, meaning each possible subset of size k has an equal probability of being chosen. To ensure the sample size does not exceed the number of available documents, we set $k = \min(k', N)$, where k' is the desired sample size. Formally, the probability of selecting a particular subset $\mathcal{D}_s$ is given by:

$$P(\mathcal{D}_s) = \frac{1}{\binom{N}{k}}, \tag{4}$$

where $\binom{N}{k}$ is the binomial coefficient representing the number of ways to choose k documents from the N documents in $\mathcal{D}$.

Stratified Random Sampling. In the preprocessing step, a random subset $\mathcal{D}_s \subseteq \mathcal{D}$ of size $|\mathcal{D}_s| = k$ is selected without replacement from the full document set $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$, where $N = |\mathcal{D}|$. To capture the diversity of the dataset, we employ a stratified random sampling approach based on a feature space $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$, where features may include document topics, lengths, or semantic embeddings. The dataset $\mathcal{D}$ is partitioned into L disjoint strata $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_L\}$ such that $\mathcal{D} = \bigcup_{l=1}^{L} \mathcal{S}_l$ and $\mathcal{S}_l \cap \mathcal{S}_{l'} = \emptyset$ for $l \neq l'$. Each stratum $\mathcal{S}_l$ contains $N_l = |\mathcal{S}_l|$ documents, with proportion $p_l = N_l/N$.

Within each stratum $\mathcal{S}_l$, a subset of $k_l = \lfloor k \cdot p_l \rfloor$ documents is sampled uniformly without replacement, ensuring the total sample size is approximately $k = \sum_{l=1}^{L} k_l$. The probability of selecting a specific subset $\mathcal{D}_s = \bigcup_{l=1}^{L} \mathcal{D}_{s,l}$, where $\mathcal{D}_{s,l} \subseteq \mathcal{S}_l$ and $|\mathcal{D}_{s,l}| = k_l$, is given by:

$$P(\mathcal{D}_s) = \prod_{l=1}^{L} \frac{1}{\binom{N_l}{k_l}}, \tag{5}$$

where $\binom{N_l}{k_l}$ is the binomial coefficient representing the number of ways to choose k_l documents from the N_l documents in stratum $\mathcal{S}_l$.

For each sampled document $d_i \in \mathcal{D}_s$, the corresponding set of terms $g(d_i)$ is retrieved, forming the few-shot exemplars provided to the large language models. This

stratified sampling approach ensures that the prompt examples respect the context window constraints of large language models while preserving dataset variability through proportional representation of diverse document features.

Prompt Construction. Using the sampled subset $\mathcal{D}_s$ obtained from either simple or stratified random sampling, we construct prompts to extract relevant terms for ontology creation. Each document $d_i \in \mathcal{D}_s$ contains an identifier, title, and text, denoted as d_i.id, d_i.title, and d_i.text, respectively. The terms $g(d_i)$ associated with each document are retrieved from a precomputed mapping, forming few-shot exemplars. These exemplars are formatted as a sequence of document-term pairs within the prompt, enclosed in designated markers to guide the language model.

For each document in the dataset $\mathcal{D}$, a prompt is constructed by combining an example prompt, derived from the sampled subset $\mathcal{D}_s$, with the document's title and text. The prompt instructs the language model to extract all relevant terms that could form the basis of an ontology, formatted as a valid Python list of terms, e.g., $['\text{term1}', '\text{term2}']$, or an empty list $[]$ if no terms are found. The specific prompt structure is as follows:

Prompt for Ontology Term Extraction

```
{Sampled subset 𝒟_s}
[var]
Title: {title}
Text: {text}
[var]
Extract all relevant terms that could form the basis of an ontology
from the above document.
Format the output as: ['term1', 'term2', ...] from texts enclosed
in [var][var] tags.
Ensure the output is a valid Python list string, e.g., ['term1',
'term2'].
If no terms are found, return [].
Do not write ‘‘‘python.
```

Figure 1. *Prompt used for ontology term extraction from sampled subset $\mathcal{D}_s$. It directs the model to extract terms enclosed in `[var]` tags and return them as a valid Python list.*

This process ensures that the language model leverages the diversity of the sampled exemplars while respecting context window constraints, enabling robust term extraction for ontology development.

3.1.2 Type Extraction

The preprocessing and sampling procedures for type extraction follow the same methodology as the term extraction task. Specifically, documents are parsed and a representative subset $\mathcal{D}_s \subseteq \mathcal{D}$ is selected via random sampling to construct few-shot exemplars. The only difference lies in the label mapping step: instead of associating documents with ontology terms, each document $d_i \in \mathcal{D}$ is linked to its corresponding types from a predefined ontology or type schema. These type annotations are obtained using a matching function over the available structured metadata. Once the document-to-type pairs are established, prompt construction proceeds analogously to the term extraction setting, by formatting the sampled documents and their types into

in-context examples suitable for guiding the large language model in zero- or few-shot classification.

Type-to-Document Matching. Let $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ denote the set of documents, where each document d_i is a structured tuple $d_i = (id_i, t_i, x_i)$, representing the identifier, title, and text content, respectively. Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_M\}$ be the predefined set of candidate types.

We define a matching function:

$$\text{match}(\tau_j, d_i) = \begin{cases} 1, & \text{if } \tau_j \in \texttt{FullWordMatch}(t_i + \ x_i); \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where `FullWordMatch` refers to case-sensitive, full-token string search using regular expressions:

$$\texttt{FullWordMatch}(s) = \{\tau_j \in \mathcal{T} \mid \texttt{re.search}(\texttt{\textbackslash b}\tau_j\texttt{\textbackslash b}, s) \neq \emptyset\} \tag{7}$$

The resulting inverse mapping function $g : \mathcal{D} \to 2^{\mathcal{T}}$ is then defined as:

$$g(d_i) = \{\tau_j \in \mathcal{T} \mid \text{match}(\tau_j, d_i) = 1\} \tag{8}$$

To ensure one-to-one visibility in sampling and avoid repetition, a set of already matched terms is tracked during iteration:

$$\mathcal{T}_{\text{seen}} \subseteq \mathcal{T}, \quad \text{such that for each } \tau_j \in \mathcal{T}_{\text{seen}}, \text{ only one } d_i \text{ is shown.} \tag{9}$$

This process results in each term τ_j being matched to a unique document d_i where it first appears in a full-word form, ensuring unbiased, interpretable few-shot exemplars for prompt construction in the type extraction task.

Prompt Construction. Similar to the ontology term extraction task, the few-shot prompt for type extraction is constructed using a sampled subset of documents $\mathcal{D}_s \subseteq \mathcal{D}$. Each document $d_i \in \mathcal{D}_s$ is paired with its associated types (concept classes) and wrapped with custom delimiters `[var]` . . . `[var]` to scope the relevant text. The prompt explicitly instructs the language model to extract only those type mentions that can serve as ontology classes, and to format the output as a valid Python list string. The final prompt template used during inference is presented below:

Prompt for Ontology Type Extraction

```
{Sampled subset D_s}
[var]
Title: {title}
Text: {text}
[var]
Extract all relevant types mentioned in the above document that
could serve as ontology classes. Only extract types found inside
[var]...[var] tags.
Format the output as a valid Python list string, for example:
['type1', 'type2'].
If no types are found, return an empty list: [].
Do not provide any additional explanation or categorize the types.
Do not write ‘‘‘python.
```

Figure 2. *Prompt used for ontology type extraction from the sampled subset $\mathcal{D}_s$. It instructs the model to extract type-level ontology class candidates from content within `[var]` tags and return them as a Python list.*

3.1.3 Batch Prompting

Let $\mathcal{D}_{\text{test}} = \{d_1, d_2, \ldots, d_T\}$ denote the set of test documents, where each d_i consists of a title t_i and text content x_i. Unlike standard prompting where each document d_i is processed individually, we adopt a *batch prompting* strategy in which the entire set $\mathcal{D}_{\text{test}}$ is passed to the LLM as a single concatenated prompt. Formally, we define the batch prompt $\mathcal{P}_{\text{batch}}$ as:

$$\mathcal{P}_{\text{batch}} = \texttt{[Instruction]} \parallel \Big\|_{i=1}^{T} \texttt{[var]}\, t_i\, x_i\, \texttt{[var]}, \tag{10}$$

where $\parallel$ denotes string concatenation, and `[var]` markers delimit the start and end of each document block. The output of the LLM is denoted as:

$$\mathcal{Y}_{\text{batch}} = f_{\text{LLM}}(\mathcal{P}_{\text{batch}}), \tag{11}$$

where $\mathcal{Y}_{\text{batch}}$ is a flat list of terms (or types) extracted from all documents jointly. The output is parsed and stored in a plain text file, with each predicted term written on a separate line.

Batch Prompt for Ontology Term or Type Extraction

```
{Sampled subset D_s}
Based on these samples, extract all relevant terms or types from
the provided JSONL file.
Your output must be a plain .txt file. Write each extracted item
(term or type) on a separate line. The output should follow this
format:
term1(type1)
term2(type2)
term3(type3)
...
Do not include any extra formatting, explanations, or bullet points.
Only return newline-separated terms or types.
```

Figure 3. *Batch prompt used for extracting ontology terms or types from a sampled JSONL dataset subset $\mathcal{D}_s$. The model is instructed to output plain text with one term-type pair per line, without additional formatting or explanation.*

3.2 Task B - Term Typing

In this task, the objective is to identify the correct type(s) for each input term. Formally, let $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_M\}$ denote the set of ontology types extracted from the training data, and let $\mathcal{V} = \{v_1, v_2, \ldots, v_L\}$ be the set of unique ontology terms. For each type $\tau_j \in \mathcal{T}$, a set of representative terms $\mathcal{E}_j \subseteq \mathcal{V}$ is selected using the same stratified and simple sampling strategy employed in Task A.

The prompt examples are then constructed by pairing each type τ_j with a small random sample of its associated terms $\mathcal{E}_j$, ensuring that all types in $\mathcal{T}$ are represented in the prompt. For the test term $v^* \in \mathcal{V}$, the goal is to predict the correct set of types:

$$h(v^*) \subseteq \mathcal{T} \tag{12}$$

using few-shot in-context learning, where $h : \mathcal{V} \to 2^{\mathcal{T}}$ is the type prediction function induced by the LLM.

Prompt for Type Assignment to Terms

```
From the sample terms provided for each type, identify the type(s)
of the term. A term can have more than one type. Only write the
types in quotation marks, and if there is more than one, separate
them with a comma.
The types must be selected from the given list|no more, no less.
Examples:
{Sampled subset $\mathcal{D}_{\mathcal{T}}$}
```

Figure 4. *Prompt used for assigning ontology types to given terms from a sampled subset $\mathcal{D}_{\mathcal{T}}$. The model is instructed to choose only from a predefined list of types and format them using quotation marks, with multiple types separated by commas.*

3.2.1 Batch Prompting

In the batch prompting setting, rather than sending one term at a time to the language model, a complete list of test terms is given as a single input, accompanied by the pre-constructed examples. The language model is then expected to identify the appropriate types for each term in the batch. The output is expected to be a valid JSON list, where each object contains an `id` and a corresponding list of `types` associated with the term.

Prompt for Batch Type Classification

```
From the sample terms provided for each type, identify the type(s)
of the term in a second JSON file.
A term can have more than one type.
Output file must be in this format:
[
{ "id": "TT_465e8904", "types": [ "type1" ] },
{ "id": "TT_01c7707e", "types": [ "type2", "type3" ] },
{ "id": "TT_b20cb478", "types": [ "type4" ] }
]
The id must be taken from the input JSON file.
You must find the type(s) for each term in the JSON file.
Sample terms for each type are provided in a text file.
Types must be selected only from the sample list.
```

Figure 5. *Prompt used for batch classification of ontology types for terms in a JSON file. The model is guided to assign one or more types per term based on provided samples, producing structured output in JSON format using the term IDs and a fixed list of allowed types.*

3.2.2 Blind

In blind subtasks, no training samples or inclusion files are available. Instead, a text file containing the complete list of types is provided. These types are used directly within the prompt construction for the blind challenge, as shown in the example below. This approach allows the model to rely solely on the type list without being exposed to any prior example-term associations.

Prompt for Blind Type Classification

```
Identify the type(s) of the term in a second JSON file.
A term can have more than one type.
Output file must be in this format:
[
{ "id": "TT_465e8904", "types": [ "type1" ] },
{ "id": "TT_01c7707e", "types": [ "type2", "type3" ] },
{ "id": "TT_b20cb478", "types": [ "type4" ] }
]
The id must be taken from the input JSON file.
You must find the type(s) for each term in the JSON file.
Types must be selected only from the types list.
```

Figure 6. *Prompt for blind classification of ontology types for terms based on a JSON file. The model is instructed to assign one or more types per term using the term IDs, with output formatted as a JSON array and types restricted to a predefined list.*

3.3 Task C - Taxonomy Discovery

For subtask C, the objective is to identify all parent–child relationships from a given text file containing ontology terms. Unlike other tasks, only the batch prompting strategy was applicable for this task due to the nature of the relation extraction process, which makes it impractical to query the language model API separately for each possible pair. A major challenge in this task was the size of the input file, which exceeded the maximum context length of large language models. Our sampling strategies were not effective in this task. To address this limitation, we adopted a *chunking strategy*, whereby the input data was split into manageable segments that fit within the model's context window.

Furthermore, for the **OBI** dataset, where many parent and child terms exhibit token-level overlap, we employed a sentence embedding similarity comparison strategy. This technique helps capture semantic relationships even when lexical similarity is high, thereby improving the accuracy of parent-child link detection in overlapping term scenarios.

3.3.1 Chunking Strategy

Let $\mathcal{P} = \{(p_i, c_i)\}_{i=1}^{M}$ denote the full set of potential parent–child pairs extracted from the source data, where p_i is a parent concept and c_i is its corresponding child. Due to the context window limit L_{max} of a given large language model (LLM), we partition the full input $\mathcal{I}$ into disjoint chunks:

$$\mathcal{I} = \bigcup_{j=1}^{K} \mathcal{I}_j, \quad \text{with} \quad |\mathcal{I}_j| \leq L_{\text{max}}, \quad \mathcal{I}_j \cap \mathcal{I}_{j'} = \emptyset \text{ for } j \neq j'. \tag{13}$$

Each chunk $\mathcal{I}_j$ is independently submitted to the LLM to extract a local set of candidate relations:

$$\mathcal{P}_j = \text{LLM}(\mathcal{I}_j), \quad \text{for } j = 1, 2, \ldots, K. \tag{14}$$

The final output set is constructed as the union of all extracted sets, followed by deduplication:

$$\mathcal{P}_{\text{final}} = \bigcup_{j=1}^{K} \mathcal{P}_j, \quad \text{with duplicates removed.} \tag{15}$$

This process ensures that the full set of hierarchical relations is recovered as completely as possible under the constraint L_{max}.

3.3.2 Batch Prompting

For Task C, the batch prompting approach is employed with the chunking strategy described previously. Let the set of training pairs be partitioned into N disjoint chunks:

$$\mathcal{T} = \bigcup_{i=1}^{N} \mathcal{T}_i, \quad \mathcal{T}_i \cap \mathcal{T}_j = \emptyset \quad \text{for } i \neq j, \tag{16}$$

and the test data similarly partitioned into M disjoint chunks:

$$\mathcal{S} = \bigcup_{j=1}^{M} \mathcal{S}_j, \quad \mathcal{S}_j \cap \mathcal{S}_k = \emptyset \quad \text{for } j \neq k. \tag{17}$$

The batch prompt is constructed by pairing each training chunk $\mathcal{T}_i$ with each test chunk $\mathcal{S}_j$, resulting in a total of

$$N \times M \tag{18}$$

prompts to be processed.

This ensures that all possible combinations of training and test chunks are covered within the context window limitations of the language model.

Prompt for Parent-Child Relation Extraction

```
From this file, extract all parent and child relations for all
pairs like examples in JSON file.
Output file must be in this format:
[
{ "parent": "parent1", "child": "child1" },
{ "parent": "parent2", "child": "child2" },
{ "parent": "parent3", "child": "child3" },
{ "parent": "parent4", "child": "child4" }
]
You must find all parent-child pairs from the input file.
Each pair should be extracted and formatted as shown above.
```

Figure 7. *Prompt for extracting parent-child relations from a JSON file. The model is instructed to identify and output all parent-child pairs in the specified JSON format.*

3.3.3 OBI subtask

In this subtask, due to the high lexical overlap between child and parent terms in the training set, we employed a sentence embedding–based strategy. Specifically, we computed the embedding vector $\mathbf{e}_c$ for each child term using a pretrained sentence embedding model. Then, for each possible parent term p in the training set, we computed its embedding vector $\mathbf{e}_p$ and calculated the cosine similarity.

The parent term with the highest cosine similarity score was selected as the most likely parent of the given child. This method leverages semantic similarity beyond surface-level token overlap and is particularly useful in datasets with high lexical redundancy, such as OBI.

4. Experiments and Results

All the reported results were calculated using the CodaLab platform of the competition[1]. The performance was measured and reported using the F1 score, which is calculated as:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{19}$$

The experiments utilize several state-of-the-art language models, including `gemini-2.5-flash-preview-05-20` [5], `grok-3` and its lighter variant `grok-3-mini` [6], `deepseek-chat-v3-0324` [7], `gpt-4o-mini` [8], and `claude-sonnet-4` [9]. These models provide a diverse range of architectures and capabilities, and their respective citations are included to acknowledge their original contributions.

4.1 Task A - Text2Onto

As shown in Table 2, batch-prompted models (indicated by "Batch") often outperform or match non-batch models in several subtasks. Claude Sonnet 4 (Batch) and Deepseek code v3 (Batch) exhibit strong performance, particularly in the scholarly and engineering domains. Stratified sampling was applied wherever feasible to ensure balanced training and evaluation across term types and domains. Overall, the Gemini model and Claude Sonnet 4 (Batch) consistently achieve the highest F1 scores in both term and type extraction tasks.

[1] https://codalab.lisn.upsaclay.fr/competitions/23065

Table 2. *F1 scores for various models across SubTasks A1 (Term Extraction) and A2 (Type Extraction) for different domains. The best-performing model for each subtask is highlighted in bold.*

SubTask	Model	F1 Score
Ecology		
A2.1 - Type Extraction	Gemini	0.62
A2.1 - Type Extraction	Deepseek code v3	0.47
A2.1 - Type Extraction	Claude Sonnet 4 (Batch)	**0.66**
Scholarly		
A1.2 - Term Extraction	Gemini	0.42
A1.2 - Term Extraction	Deepseek code v3	0.34
A1.2 - Term Extraction	Grok 3 mini	0.39
A1.2 - Term Extraction	GPT_4o mini	0.35
A1.2 - Term Extraction	Claude Sonnet 4 (Batch)	**0.45**
A2.2 - Type Extraction	Gemini	0.47
A2.2 - Type Extraction	Deepseek code v3	0.43
A2.2 - Type Extraction	Claude Sonnet 4 (Batch)	0.61
A2.2 - Type Extraction	Grok 3 (Batch)	**0.65**
Engineering		
A1.3 - Term Extraction	Gemini	0.57
A1.3 - Term Extraction	Deepseek code v3	0.58
A1.3 - Term Extraction	GPT_4o mini	0.51
A1.3 - Term Extraction	Claude Sonnet 4 (Batch)	0.45
A1.3 - Term Extraction	Deepseek code v3 (Batch)	0.57
A1.3 - Term Extraction	Deepseek + Claude (Batch)	0.58
A2.3 - Type Extraction	Grok 3 (Batch)	**0.62**
A2.3 - Type Extraction	Gemini	0.32
A2.3 - Type Extraction	Deepseek code v3	0.18
A2.3 - Type Extraction	Claude Sonnet 4 (Batch)	**0.66**
A2.3 - Type Extraction	Grok 3 (Batch)	0.30

4.2 Task B - Term Typing

Table 3. *F1 scores for various models on Term Typing subtasks (Task B).*

SubTask	Model	F1 Score
OBI		
B1 - Term Typing	Deepseek code v3	0.75
B1 - Term Typing	Gemini	0.72
B1 - Term Typing	GPT_4o mini	0.56
B1 - Term Typing	Claude Sonnet 4 (Batch)	**0.94**
MatOnto		
B2 - Term Typing	Deepseek code v3	0.38
B2 - Term Typing	Gemini	0.41
B2 - Term Typing	GPT_4o mini	0.44
B2 - Term Typing	Claude Sonnet 4 (Batch)	**0.57**
SWEET		
B3 - Term Typing	Deepseek code v3	0.44
B3 - Term Typing	Gemini	0.48
B3 - Term Typing	GPT_4o mini	0.28
B3 - Term Typing	Claude Sonnet 4 (Batch)	**0.69**
Blind		
B4 - Term Typing	Claude Sonnet 4 (Batch)	**0.76**
B5 - Term Typing	Claude Sonnet 4 (Batch)	**0.93**

As illustrated in Table 3, models utilizing batch prompting (labeled "Batch") generally outperform or closely rival their non-batch counterparts across the Term Typing subtasks. Claude Sonnet 4 (Batch) consistently delivers top F1 scores across all domains, including both standard and blind test sets.

4.3 Task C - Taxonomy Discovery

Table 4. *F1 scores for various methods on Taxonomy Discovery subtasks (Task C). Batch-prompted methods are indicated accordingly.*

SubTask	Method	F1 Score
OBI		
C1 - Taxonomy Discovery	F1 token overlap	0.34
C1 - Taxonomy Discovery	Bge-m3	**0.35**
C1 - Taxonomy Discovery	All-mpnet-base-v2	**0.35**
C1 - Taxonomy Discovery	All-MiniLM-L6-v2	**0.35**
C1 - Taxonomy Discovery	NovaSearchstella_en_1.5B_v5	**0.35**
MatOnto		
C2 - Taxonomy Discovery	Claude Sonnet 4 (Batch)	**0.66**
C2 - Taxonomy Discovery	Grok 3 (Batch)	0.37
SWEET		
C3 - Taxonomy Discovery	Claude Sonnet 4 (Batch)	**0.5**
SchemaOrg		
C5 - Taxonomy Discovery	Claude Sonnet 4 (Batch)	**0.66**
PO		
C8 - Taxonomy Discovery	Grok 3 (Batch)	**0.27**

As shown in Table 4, performance on the OBI subtask indicates only marginal differences between the simple token overlap method and advanced sentence embedding techniques. Despite employing more sophisticated encoders such as `bge-m3` [10], `all-mpnet-base-v2` [11], `all-MiniLM-L6-v2` [12], and `NovaSearchstella_en_1.5B_v5` [13], F1 scores remain tightly clustered around the token overlap baseline. This suggests that for structured domains like OBI, token-level similarity still provides competitive results. However, in other subtasks such as MatOnto and SchemaOrg, batch-prompted methods like `Claude Sonnet 4 (Batch)` show considerable performance gains.

5. Conclusions

This study demonstrates that effective prompt engineering strategies can mitigate the inherent limitations of LLMs arising from their restricted context windows. By employing such strategies, it is possible to utilize LLMs efficiently, circumventing the need for costly hardware and extensive, time-consuming training procedures. Coupled with the capabilities of contemporary powerful LLMs, these methods facilitate the fully automatic construction of high-quality ontologies. Furthermore, our findings indicate that sentence embedding–based approaches do not significantly outperform simple F1 token overlap metrics, with similar results observed across a spectrum of pretrained models ranging from 1.5 billion to 0.1 billion parameters. These insights highlight promising directions for leveraging LLMs in knowledge representation tasks while optimizing resource usage.

Data availability statement

All data used in this paper was based on the training data provided by the task organizers.

Underlying and Related Material

The code used in this work will be made available after the competition at the following repository: https://github.com/rarahnamoun/LLMs4OL-Challenge-ISWC-2025

Author contributions

Rashin Rahnamoun: Conceptualization, Methodology, Software, Validation, Investigation, Data Curation, Writing - Original Draft.
Mehrnoush Shamsfard: Writing - Review & Editing, Supervision.

Competing interests

The authors declare that they have no competing interests.

References

[1] H. B. Giglou, J. D'Souza, and S. Auer, "Llms4ol: Large language models for ontology learning", *International Semantic Web Conference*, vol. 2023, pp. 408–427, 2023.

[2] H. B. Giglou, J. D'Souza, and S. Auer, "Llms4ol 2024 overview: The 1st large language models for ontology learning challenge", *arXiv preprint arXiv:2409.10146*, 2024.

[3] H. B. Giglou, J. D'Souza, S. Sadruddin, and S. Auer, "Llms4ol 2024 datasets: Toward ontology learning with large language models", *Open Conference Proceedings*, vol. 4, pp. 17–30, 2024.

[4] H. Babaei Giglou, J. D'Souza, N. Mihindukulasooriya, and S. Auer, "Llms4ol 2025 overview: The 2nd large language models for ontology learning challenge", *Open Conference Proceedings*, 2025.
[5] "Google/gemini-2.5-flash-preview-05-20", Accessed: Jul. 7, 2025. [Online]. Available: https://gemini.google.com/app.
[6] "X-ai/grok-3", Accessed: Jul. 7, 2025. [Online]. Available: https://grok.com/.
[7] "Deepseek/deepseek-chat-v3-0324", Accessed: Jul. 7, 2025. [Online]. Available: https://huggingface.co/deepseek-ai/DeepSeek-V3.
[8] "Openai/gpt-4o-mini", Accessed: Jul. 7, 2025. [Online]. Available: https://platform.openai.com/docs/models/gpt-4o-mini.
[9] "Anthropic/claude-sonnet-4", Accessed: Jul. 7, 2025. [Online]. Available: https://www.anthropic.com/claude/sonnet.
[10] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation", *Findings of the Association for Computational Linguistics: ACL 2024*, vol. 2024, pp. 2318–2335, Aug. 2024. DOI: 10.18653/v1/2024.findings-acl.137.
[11] "All-mpnet-base-v2", Accessed: Jul. 7, 2025. [Online]. Available: https://huggingface.co/sentence-transformers/all-mpnet-base-v2.
[12] "All-minilm-l6-v2", Accessed: Jul. 7, 2025. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.
[13] D. Zhang, J. Li, Z. Zeng, and F. Wang, "Jasper and stella: Distillation of sota embedding models", *arXiv preprint arXiv:2412.19048*, Dec. 2025. [Online]. Available: https://arxiv.org/abs/2412.19048.