Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Algorithms for extraction of definitions in laws

Bc. Jan Hošťálek

Supervisor: Ing. Petr Křemen, Ph.D.
Field of study: Open Informatics
Subfield: Data Science
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Hošťálek Jan**	Personal ID number:	**492268**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Data Science**		

II. Master’s thesis details

Master’s thesis title in English:

Algorithms for extraction of definitions in laws

Master’s thesis title in Czech:

Algoritmy pro extrakci definic v právních předpisech

Guidelines:

The goal of the work is to design, implement and test algorithms for extracting formal term definitions from Czech laws and build a vocabulary of such terms.
1. Become familiar with the structure of semantic web standards (RDFS,OWL,SPARQL), especially focusing on modeling OWL ontologies
2. Explore current NLP techniques for extracting structured ontological descriptions and definitions, assess their suitability for the Czech language.
3. Design novel techniques for (I) detecting and extracting definitions from legal acts (preferably Czech), (II) extracting formal OWL representation from the detected definitions. Make the design configurable and parameterizable.
4. Test on a suitable set of legal documents. Showcase their usage in OWL ontologies and assess the quality of all developed techniques.

Bibliography / sources:

[1] OWL 2 Web Ontology Language: Primer (Second Edition) Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph, eds. W3C Recommendation, 11 December 2012, http://www.w3.org/TR/2012/REC-owl2-primer-20121211/. Latest version available at http://www.w3.org/TR/owl2-primer/.
[2] Hassanpour, Saeed & O'Connor, Martin & Das, Amar. (2011). A Framework for the Automatic Extraction of Rules from Online Text. 266-280. 10.1007/978-3-642-22546-8_21.
[3] Judkins J, Utecht J, Brochhausen M. Easy Extraction of Terms and Definitions with OWL2TL. CEUR Workshop Proc. 2016 Aug;1747:D205. PMID: 28035214; PMCID: PMC5189984.
[4] Stevens, Robert & Malone, James & Williams, Sandra & Power, Richard & Third, Allan. (2011). Automating Generation of Textual Class Definitions from OWL to English. Journal of biomedical semantics. 2 Suppl 2. S5. 10.1186/2041-1480-2-S2-S5.
[5] Saeeda, L., Med, M., Ledvinka, M., Blaško, M., Křemen, P. (2020). Entity Linking and Lexico-Semantic Patterns for Ontology Learning. In: Harth, A., et al. The Semantic Web. ESWC 2020. Lecture Notes in Computer Science(), vol 12123. Springer, Cham. https://doi.org/10.1007/978-3-030-49461-2_9

Name and workplace of master’s thesis supervisor:

Ing. Petr Křemen, Ph.D. Knowledge-based Software Systems FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **25.01.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Petr Křemen, Ph.D.	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature		Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to express my gratitude to the thesis supervisor Ing. Petr Křemen, Ph.D. for his guidance, valuable advice and suggestions.

Declaration

I declare that I have written the submitted project independently and that I have listed all the information sources used in accordance with the Methodological Guideline on the Observance of Ethical Principles in the Preparation of University Theses.

Prague, 24 May 2024

Jan Hošťálek

Abstract

This work presents new algorithms and approaches for extracting definitions from Czech legislation documents using advanced Semantic Web technologies and Large Language Models. It focuses on the applicability of language models to the tasks of term extraction, term definitions and subsequent ontology creation. From a methodological perspective, it includes techniques such as In Context Learning and Retrieval Augmented Generation, leading to improved extraction and interpretation of legal terminology. The findings offer important insights into the applicability of language models to specific NLP tasks in the area of legal document analysis and processing, which is related to the creation of knowledge ontologies.

Keywords: Term Extraction, Definition Extraction, Ontology Creation, NLP, LLM, Few Shot RAG, Embeddings, ICL

Supervisor: Ing. Petr Křemen, Ph.D.

Abstrakt

Tato práce představuje inovativní algoritmy a přístupy pro extrakci definic z českých legislativních dokumentů s využitím technologií sémantického webu a velkých jazykových modelů. Zaměřuje se na použitelnost jazykových modelů pro úlohy extrakce termínů, definic termínů a následné tvorby ontologií. Z metodologického hlediska zahrnuje techniky jako In Context Learning a Retrieval Augmented Generation, které vedou ke zlepšení extrakce a interpretace právní terminologie. Výsledky obsahují důležité poznatky o použitelnosti jazykových modelů pro konkrétní NLP úlohy v oblasti analýzy a zpracování právních dokumentů, které souvisejí s tvorbou znalostních ontologií.

Klíčová slova: Extrakce termínů, extrakce definic, tvorba ontologií, NLP, LLM, Few Shot RAG, Embeddings, ICL

Překlad názvu: Algoritmy pro extrakci definic v právních předpisech

Contents

Figures

Tables

Chapter 1

Introduction

We present and benchmark novel methodologies for extracting terms and definitions from Czech legislative documents, which we subsequently use for building ontologies, leveraging the capabilities of advanced semantic web technologies and large language models. Our approach involves the adaptation and enhancement of existing algorithms and methodologies, incorporating state-of-the-art language embeddings and language model techniques.

1.1 Motivation

The motivation behind this work is the integration and use of large language models for ontology population. The biggest language models have surpassed everyone's expectations and have outperformed previous state-of-the-art methods in many areas of natural language processing in the last months/years. One of the side objectives of this work is to explore and benchmark the applicability of large language models to the creation of complex fully interpretable robust knowledge bases. This can be used in the future to potentially improve the quality of learning/fine-tuning large language models that are currently being primarily trained on huge text corpora.

However, it is the characteristics of the text corpus itself and its weaknesses that introduce a very strong bias in the subsequent inference with respect to the input data. For example, if the model was trained on a dataset that was skewed towards a recently very popular phenomenon, fake news, it could infer texts containing concepts that contradict each other [1].

Our work focuses on the controlled semi-automated structuring of information provided in input text documents, specifically the laws of the Czech Republic. Structuring means extraction and subsequent creation of knowledge - linked databases - ontologies.

1.2 Streamlining and Making Legislation Accessible

Currently, a large portion of Czech legislation is undergoing multiple waves of digitalization under the initiative *Digitální Česko*[1] This initiative aims to create, preserve, mediate, and maintain legislative documents digitally. Most digitalized legal documents are commonly in PDF format or available through web portals.

Currently, the web portals of the ministries themselves or those managed by private entities or communities are the most popular sources/access points to laws, ordinances, etc. Other kinds of legal documents include decrees, regulations, directives, and resolutions. The characteristics of input pure legal text documents from these web sources often include complex language, legal jargon, and detailed provisions that require accurate interpretation.

One of the most visited portals, which is also the one that we have used in our work to obtain the text of specific laws, is *www. zakonyprolidi. cz.* The main goal of these web interfaces is to provide a simple and quick insight into the desired legislative documents, which they do very quickly and reliably. The speed of finding indexed parts of the law based on a user's full-text query is a key advantage of these systems compared to physical copies or persistently stored files.

Despite the fact that access to usually complex and often widespread sources has significantly advanced recently, it is necessary to mention several main disadvantages and define areas for improvement both in mediation and in subsequent forms of searching for information. Let's define areas of improvement with list of reasons why should we focus on structuralizing information into connected ontology[2] systems:

[1]https://digitalnicesko.gov.cz/

[2]Ontology is a structured framework for organizing information that defines a set of concepts and the relationships between them within a particular domain of knowledge.

- Standardization of Formats: Ontologies provide a unified and consistent way to define and store information. This standardized format includes not only the legal terms themselves but also the relationships between them, which is crucial for understanding the context and dependencies in legal text. Standardization simplifies the exchange and sharing of data between different systems and platforms.
- Linking with Other Ontologies: Ontologies allow for the creation of links between different knowledge areas. For example, an ontology of Czech law can be linked with ontologies from the fields of economics, healthcare, or education, enabling complex analyses and better understanding of the connections between different legislations and sectors.
- Integration and Information Retrieval: Thanks to their standardization[34] and the characteristics of linked data, ontologies can easily be integrated into a wide range of information systems, including legislative search engines. Ontologies can also be connected to automated decision-making processes and systems, which can enhance the efficiency of providing legal services to customers. Besides searching and informed decision-making, ontologies can be used for automatic document generation or even automatic notification of relevant legislative changes.
- Supporting Decision Processes and Compliance: Structured and well-connected legal information enables organizations to better navigate legal requirements and enhances their ability to comply with regulatory demands. For instance, an ontology can automatically identify areas where regulations might be violated and suggest corrective measures.
- Simplifying Updates and Maintenance: Given that legal regulations are constantly changing and evolving, ontologies facilitate easier updates and maintenance of databases, as changes in one document or segment can be automatically reflected across all relevant areas of the system.

1.3 Legal Term

The building blocks of all laws are legal terms that represent domain-specific legal principles, processes, rights, obligations and entities. Their recognition and correct understanding is crucial for subsequent interpretation, as they precisely define the legal norms that regulate the behaviour and relations of individuals to the state. Recognizing these entities is the first step to successfully structuring entire documents.

Now that we have a basic idea of what Legal Term means semantically, let's define its syntactic form and representation for the purposes of our experiments and algorithms. First of all, the stylistics in which each legal document is written is different. In other words, an object that a domain expert considers as a Term may be stated, for example, in the nominative, instrumental or genitive case. This is because Czech, a Slavic language, is characterized by a rich inflectional system applied to nouns, verbs, and adjectives. This inflection influences the syntactic and semantic structure of legal terms and allows them to appear in different forms depending on grammatical gender, number, genus, and verbal aspect. Additionally, Czech language's semantic richness, where multiple words can exist for similar things and a single word can have multiple meanings, along with common typos, pose significant challenges for automated text processing. These variations and complexities, although semantically consistent, complicate the extraction of legal terms from documents for analysis and application. We propose multiple approaches for infinitive state conversion later in Chapter 5.1.4. A term in the infinitive form, whether it is a single-word or multi-word phrase, represents a class in the ontology. In addition to the identifier (e.g., *dopravní-infrastruktura* transport-infrastructure), which may include an optional definition, we also aim to extract candidates for terms that are not defined in the text.

1.4 Definition of Legal Term

The secondary principal objective of our work was to propose methodologies for extracting definitions associated with the previously identified terms. The definition of a specific term refers to the segment of the original text that explicitly captures and defines the term's meaning within a particular domain, area of law, and, most critically, within the specific phrasing of the legislation. It is important to note that the precise definition of the same term may vary across different legal texts. Additionally, for the purposes of this study, we do not consider indirect definitions or references to definitions of the same terms in other legislative documents as valid definitions. Example of direct *"Svéprávnost je způsobilost nabývat pro sebe vlastním právním jednáním práva a zavazovat se k povinnostem (právně jednat)."* vs indirect *"Pokud se v tomto zákoně používá pojmu stavba, rozumí se tím podle okolností i její část nebo změna dokončené stavby."*.

Chapter 2

Methods and Technologies

In this chapter, we explore the various methods and technologies employed in our work. Each section provides an overview of the specific technologies and methodologies that are relevant to the tasks of extracting terms, definitions, and building ontologies from Czech legislative documents. The sections are organized to first introduce the foundational technologies and then move to more advanced techniques and their applications.

2.1 Semantic Web Technologies

2.1.1 RDF(S)

The Resource Description Framework (RDF) is a standard model for data interchange on the web [35]. RDF extends the linking structure of the web to use URIs to name the relationship between things as well as the two ends of the link. This simple model allows structured and semi-structured data to be mixed, exposed and shared across different applications.

RDF Schema (RDFS) is designed to define schemas over RDF graph data [33]. Its main purpose is to specify what a class is and to determine the domain and range of properties. This provides a structured framework for describing groups of related resources and the relationships between them.

As a semantic extension of RDF, RDFS plays a critical role in creating vocabularies and defining ontological structures essential for the Semantic Web.

2.1.2 OWL2

Web Ontology Language (OWL) is a comprehensive framework for modelling complex knowledge structures. In OWL, entities are identified using IRIs. An ontology is also uniquely identified using an ontology IRI (with versions). The OWL language allows importing other ontologies, thus enabling their interconnection and enrichment.

```
Class: :FatherOfSons
    SubClassOf: :hasChild some owl:Thing and :hasChild only :Man.
```

Figure 2.1: Manchester Syntax Example

OWL provides different syntaxes, including DL, Manchester (Figure 2.1), and Turtle (Figure 2.2) syntaxes, each serving specific use cases and preferences. Custom annotations can be assigned to various resources, such as classes, properties, and axioms, enhancing the expressiveness of the language.

```
:FatherOfSons rdf:type owl:Class ;
    rdfs:subClassOf [ rdf:type owl:Class ;
    owl:intersectionOf ( [ rdf:type owl:Restriction ;
                           owl:onProperty :hasChild ;
                           owl:someValuesFrom owl:Thing]
                         [ rdf:type owl:Restriction ;
                           owl:onProperty :hasChild ;
                           owl:allValuesFrom :Man ] )
```

Figure 2.2: OWL/RDF serialization in Turtle Example. Source:[10]

It also has diverse property expressions. Object Properties in OWL support characteristics like Functional, Irreflexive, Asymmetric, etc., and they can be defined in terms of domains, ranges, subproperties, and more. Similarly, Data Properties in OWL define data properties with characteristics and relationships like domains and subproperties. Object properties connect individual entities (e.g., a person to their parent), whereas data properties link individuals to specific data (e.g., a person to their birthdate).

OWL supports restriction to custom data ranges with standard set operations. This enables the modelling of specific data characteristics and constraints. Classes can be defined in terms of their relationships with other classes, such as subclasses, equivalent classes, disjoint classes, and keys, which help in creating structured and well-defined class hierarchies.

The OWL language family includes OWL Full, OWL 2 DL, and subsets like OWL 2 EL, QL, and RL, each with specific features and applications. OWL 2 EL is optimized for rich class taxonomies, OWL 2 QL for large datasets, and OWL 2 RL offers a rule-based semantic approach.

OWL and its extension OWL 2 offer a powerful and flexible framework for representing complex knowledge about entities, their interrelations, and their properties. It supports a range of logical constructs, data types, and syntactic features, making it suitable for creating detailed, rich ontologies in various domains. The language's design ensures that it can satisfy diverse modelling needs while maintaining logical consistency and providing mechanisms for efficient reasoning. The process of building ontologies is a task that requires knowledge of both the language itself and the domain we are modelling. Our work aims to propose algorithms and methods to facilitate and automate the creation of OWL ontologies focused on Czech legal structures using modern NLP methods.

2.2 Large Language Models

Natural Language Processing (NLP), one of the main branches of Artificial Intelligence (AI), has evolved significantly in the last two decades, moving from traditional methods based on Statistical Language Models (SLM) to Neural Language Models (NLM), specifically most commonly the use of Recurrent Neural Networks (BiLSTM)[41]. Recently, however, the field has experienced another revolution when pre-trained language models (PLMs) using the easily parallelizable Transformer architecture combined with training over large text corpora, BERT [7]. Subsequent experiments with the architecture and pre-training strategies (Roberta [13] & GPT2 [26]) have demonstrated the ability to generalize knowledge and the possibility of solving other tasks using fine-tuning in addition to the task solved during pre-training. Based on further research and scaling law [9], models that we refer to as Large Language Models (LLM) with billions of parameters were pre-trained on large corpora using massive distributed computing power.

In our work, we propose solutions to individual tasks using the ability of LLM In-context Learning in combination with Retrieval Augmented Generation.

2.2.1 In-context Learning (ICL)

Leveraging general knowledge of the largest models for specific language processing tasks usually requires model alignment. In the case of using open-source models that have publicly available weights, we can use a fine-tuning approach.

For closed-source models that we cannot fine-tune, alternative alignment approaches have been presented [3], without the need for top k layer re-learning (gradient updates). One such approach is In-context learning, which leverages the context window size and is based on injecting rules, and instructions enriched with task demonstrations (few shot examples) into the prompt.

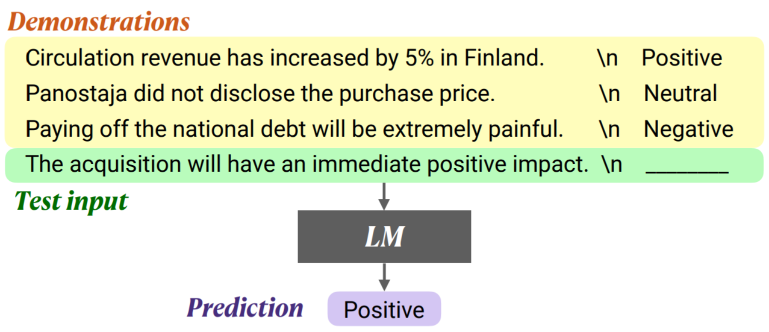

Demonstrations
Circulation revenue has increased by 5% in Finland. \n Positive
Panostaja did not disclose the purchase price. \n Neutral
Paying off the national debt will be extremely painful. \n Negative
The acquisition will have an immediate positive impact. \n ______
Test input
LM
Prediction
Positive

Figure 2.3: In context learning demonstration consists of k input-label pairs from the training data (In this case $k = 3$)[21].

This approach does not require any fine-tune computing power and it has been shown, that for specific tasks it achieves similar performance as fine-tuned versions of the same model [6].

Recent research indicates that learning in context acts as an implicit form of fine-tuning [6]. It's suggested that Transformer attention mechanisms perform a kind of dual gradient descent during In-Context Learning (ICL), where Large Language Models (LLMs) generate meta-gradients from demonstration examples. These meta-gradients adjust the original model weights, leading to a meta-tuned model. Comparative studies between learning-in-context and explicit fine-tuning on real tasks have revealed similarities, supporting the concept of ICL as a form of implicit fine-tuning.

The importance of correct in-context demonstrations

Recent experiments have revealed that the exactness of ground truth in demonstrations is not as crucial for effective ICL[21]. Replacing labels in demonstrations with random ones only marginally affects LLMs' performance in classification and multi-choice tasks. This highlights the importance of other elements in demonstrations, such as the label space, input text distribution, and overall format, in guiding LLMs during ICL.

These findings are particularly important for our subtasks. For example, in the initial phase of the ontology-building process, we can use non-domain-specific demonstrations without significant performance hit, and later use the revised knowledge by a domain expert to further improve the performance.

2.3 Entity Similarity Measures

Entity similarity measures are pivotal in ontology building, especially for tasks like entity linking. These measures enable the comparison of different entities, facilitating their accurate categorization and linkage.

2.3.1 Classical Methods

Classical methods, such as the Hamming distance, have long been employed in entity similarity assessments. These methods are foundational, offering simple yet effective ways to measure differences between entities based on their features.

- **Hamming Distance:** It measures the number of positions at which the corresponding symbols are different, typically used for strings of equal length. Given two strings of the same length A and B, the Hamming Distance $H(A, B)$ is defined as:

$$H(A, B) = \sum_{i=1}^{n} [A_i \neq B_i] \tag{2.1}$$

- **Jaccard Similarity:** It evaluates similarity and diversity by comparing the size of the intersection and the union of sample sets. For two sets A and B, the Jaccard Similarity $J(A, B)$ is calculated as:

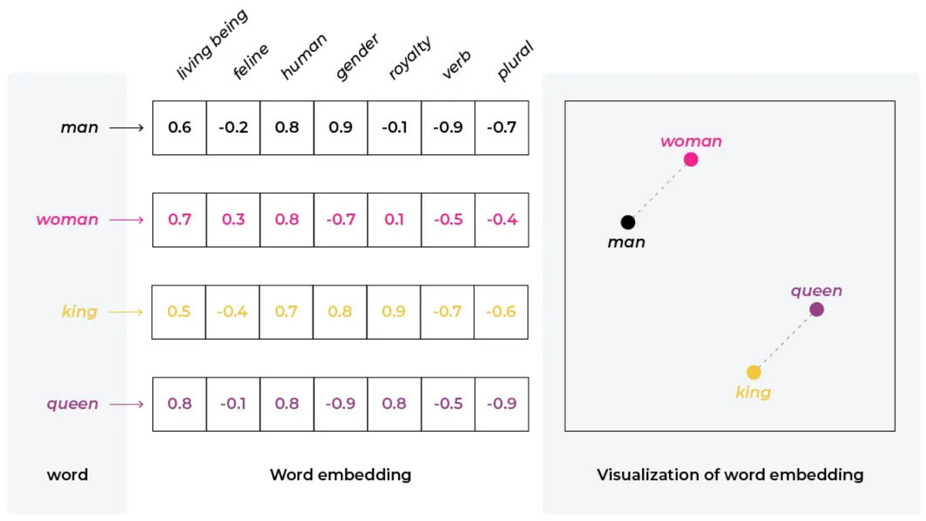
  $$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.2}$$

  Note that the element of the set can be arbitrary, e.g. single letter, n-gram or token.

- **Levenshtein Distance:** Useful for measuring the difference between two sequences by quantifying the minimum number of single-character edits required to change one word into the other.

2.3.2 Embeddings

Embedding-based methods have significantly changed the way we process, represent, and most importantly compare words, phrases, sentences, and even entire chunks of text. By representing entities as vectors in a continuous vector space, these methods excel at capturing the complex semantic and syntactic relationships that exist within the language by creating high-dimensional clusters. Unlike classical methods, embeddings computed using deep unsupervised learning offer an interpretable, easily comparable representation of language elements.



word	living being	feline	human	gender	royalty	verb	plural
man	0.6	-0.2	0.8	0.9	-0.1	-0.9	-0.7
woman	0.7	0.3	0.8	-0.7	0.1	-0.5	-0.4
king	0.5	-0.4	0.7	0.8	0.9	-0.7	-0.6
queen	0.8	-0.1	0.8	-0.9	0.8	-0.5	-0.9

Figure 2.4: Comparison of word embeddings in tabular form and their 2 principal components capturing the highest variance (PCA), demonstrating how words with similar semantic attributes cluster together in reduced dimensional space [4].

State-of-the-Art Embeddings

The latest advancements in embeddings, particularly in multilingual contexts, have been crucial in enhancing similarity measures. State-of-the-art models like ada002 offer robust capabilities in processing and understanding multiple languages, making them invaluable for many tasks such as entity linking. They excel in capturing contextual meanings and subtle linguistic variations across different languages. However, these models also come with their own set of challenges, including the need for extensive computational resources - usually trained on extensive text corpora, potential biases in training datasets, and complexities in their interpretability.

Word2Vec

The idea behind word2vec is one of the most fundamental in the field of semantic word representation. Its approach of embedding words in a high-dimensional vector space depending on their context opened up new ways of understanding and measuring the semantics of language [20][19]. Even though state-of-the-art methods have now surpassed this approach, the background idea is still the same, and its reference in many modern tools, such as the Stanza NLP tool that we use in our work [25], confirms its versatility and effectiveness.

2.4 Preprocessing

One of the main tasks and objectives of this thesis is to improve the quality of pre-processing of the input text, which is in the form of Czech law documents. We have explored and compared state-of-the-art natural language preprocessing methods to solve the individual tasks described in Chapter 3. These include tokenization, lemmatization, part-of-speech recognition (POS) and named entity recognition (NER).

2.4.1 Part-of-Speech Tagging

Part-of-Speech (POS) tagging is one of the fundamental tasks in Natural Language Processing, which consists of matching word types (e.g., noun, verb, adjective, adverb, etc.) to individual words in the original text based on the form and context of the word. This process is essential for further linguistic analysis such as partial parsing, information extraction, question answering and other NLP-based applications [14].

Commonly used extraction methods can be based on many principles. Classical methods are often based on rules or statistical models. Rule-based methods use a set of predefined rules to categorize words into parts of speech based on their morphological, syntactic and contextual properties. These rules can be language-specific and often require deep knowledge of grammar and syntax.

Another common approach is to use statistical methods that learn probabilistic models from large amounts of textual data. These models can be based on various machine learning techniques such as hidden Markov models (HMMs) or transformation-based taggers[14].

In addition to the aforementioned approaches, models based on deep neural networks have begun to dominate this field with state-of-the-art results. Since the design and learning of POS tag models for the Czech language is not the main goal of our work, we chose to use and compare the following approaches. The first, which is currently used within the TermIT ecosystem, is Morphodita [32].

Morphodita

MorphoDiTa is a toolkit that provides morphological analysis of inflectional languages[1], mainly Czech, by automatically clustering word forms into morphological templates [32]. This is achieved without the need for linguistic knowledge by using a sorting-based method that identifies common stems and generates templates from word endings. Building a morphological dictionary in combination with supervised averaging of perceptron taggers effectively

[1]Inflectional language changes the form or ending of some words when the way in which they are used in sentences changes: Latin, Polish, Finnish or Czech are all highly inflected languages.

determines the possible lemma-tagger pairs. The main advantage is fast processing and low resource requirements.

Stanza

On the other hand, multilingual (including Czech) state-of-the-art results were achieved by Stanford's NLP Group tool called Stanza, which uses their UD Models[25]. More specifically their Part-of-speech tagger utilizes a highway Bidirectional Long Short-Term Memory (BiLSTM) neural network with inputs from 1) word embeddings (word2vec[19] & fasttext[2]) 2) frequent word embedding, for all words that occurred at least seven times in the training set 3) character-level embedding, generated from a unidirectional LSTM over characters in each word.

2.4.2 Named Entity Recognition

Named Entity Recognition (NER) is one of the fundamental NLP tasks that focuses on identifying and classifying words or phrases in source text into predefined categories such as names, organizations, places, time elements, counts, monetary values, and more. In other words, this process involves extracting structured information from unstructured text, which allows transforming the raw data into a more understandable and analyzable format. Even state-of-the-art out of the box solutions, such as the aforementioned

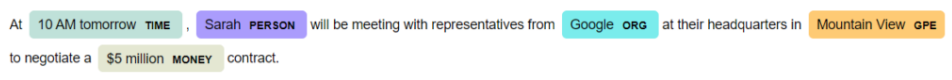

At 10 AM tomorrow TIME , Sarah PERSON will be meeting with representatives from Google ORG at their headquarters in Mountain View GPE to negotiate a $5 million MONEY contract.

Figure 2.5: Example showing a sentence highlighted with Named Entities[37].

MorphoDiTa [32], are not applicable for the purpose of finding Term Candidates in Czech legislative because in most cases the term does not belong to any of the pre-defined categories of the model. In our case, we would need to train our own NER model for which we would need to define individual classes whose superior meaning would be Legal Term. So the output would have as a term both e.g. 'budova' ('building'), which is a physical object, and for example 'svéprávnost' ('capacity'[2], which is a term in no way semantically similar to the term building.

[2]Legal capacity refers either to the legal capacity of a person to have rights and liabilities or to the very personality of an entity other than a natural person.

Candidate Entity Set Generation and Scoring

Generating and scoring a set of candidate entities in the context of entity and lexical-semantic pattern linking primarily involves two key steps: 1) generating a set of candidate entities based on string matching between the mention of the entity in the text and the representation of similar entities in the knowledge base, and 2) defining a similarity score for each candidate entity. This process often uses techniques such as surface form expansion to identify different variants of an entity, such as abbreviations, that are contextually relevant to the document being processed [28]. However, a significant drawback of this approach is its inconsistent performance across domain-specific datasets and its direct dependence on the quality and scale of the knowledge base.

Few Shot RAG for domain-specific Named Entity Recognition

The Few Shot Retrieval-Augmented Generation (RAG) approach for specific Named Entity Recognition (NER), represents a significant advance in the field of text analysis. This technique adapts the general RAG framework [11] to work efficiently with a minimum of training data, which is a crucial feature given the specialized nature of some domains where large annotated datasets may not be available [36].

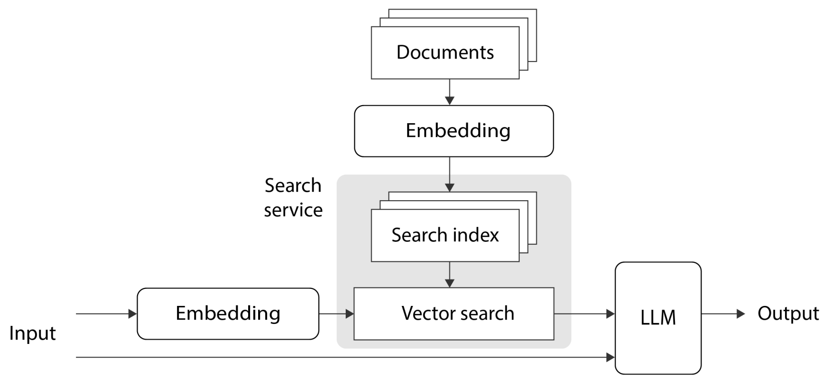


Figure 2.6: High-level overview of RAG with vector search[30].

In our case Few Shot RAG is adapted to recognize and classify specific entities in text, such as Terms and their Definitions. The input is combination of snippet of Legal Document and few examples demonstrating correctly tagged Terms in input text. The output demonstrates Few Shot RAG's ability to accurately identify and annotate these specific entities in the text. This approach not only increases the efficiency of specialized text processing, but also improves the accuracy of entity recognition in contexts where common NER models may fail due to the unique linguistic characteristics of domain-specific language. In addition, the simplicity of this approach allows to bias the extraction towards more satisfactory and precise results by using specific examples based on the underlying law document.

For example, when evaluating this approach on construction law documents, we were able to achieve better results by embedding examples of correctly identified candidate terms specifically from the construction industry in the LLM context. The example input/output of this method can be visualized in Figure 2.7.

```
Pro účely tohoto nařízení se rozumí:
a) areálem část území nečleněná veřejnými prostranstvími,
b) blokem ucelená část území, tvořená souborem pozemků,

Pro účely tohoto nařízení se rozumí:
a) [TERM:areálem] část území nečleněná veřejnými prostranstvími,
b) [TERM:blokem] ucelená část území,
```

Figure 2.7: Tagged term candidates within the context of the regulatory framework set by the Czech Telecommunication Office [5].

2.4.3 Lemmatization

Based on recent breakthroughs in rich context approaches significant improvements were achieved as the algorithms began to better understand context and morphology, leading to more accurate lemmatization. This was especially evident in languages with rich inflectional morphology, where the context and POS of a word significantly affect its lemma - Czech.

Lemmatization, the process of reducing words to their base or dictionary form (lemma), is crucial for establishing the consistency and semantic accuracy of elementary entities within the ontology. Properly formalized classes and properties enable and simplify the extensibility and linking of ontologies. However, while building an ontology, especially in legal domains, we often encounter multi-word terms (MWTs) - phrases, that often cannot be adequately expressed in one-word terms. For example, *plná moc* or *práva duševního vlastnictví*. These terms have a specific, legally defined meaning that is not simply the sum of their parts (*právo duševní vlastnictví*). Traditional lemmatization approaches process the input text at the level of individual words, which means that they fail to capture the semantics and therefore are unable to correctly lemmatize the multi-word term.

To address these problems, there is a growing need not only for lemmatization but also for POS tagging and NER approaches that are specifically tailored to work with MWTs. Such systems require a deeper level of syntactic and semantic analysis capable of understanding the nuances of entities at the phrase level. In this respect, advanced NLP techniques, including context-aware neural models and algorithms designed to recognize and process MWEs, are essential. These models use larger contextual windows and syntactic parsing to better capture the relationships between words in a phrase, ensuring that multi-word legal terms are accurately identified and lemmatized as a whole, rather than as disjointed individual words.

In our work, we evaluated the aforementioned approach of Few Shot RAG tailored to the task of candidate term extraction, including MWTs. We designed and formulated domain-specific instructions and examples of correctly lemmatized candidates. Which allowed us to achieve results presented in Chapter 4.2. Please note the difference: Lemmatization considers the context and converts the word *better* to its meaningful base form *good*, which is called Lemma. On the other hand, Stemming is a process that stems or removes the last few characters from *running*, often leading to incorrect meanings and spelling, such as *runn*.

2.5 LLM Agents and Planners

Even though the fundamental methods presented for utilizing language models in our task of extracting terms and definitions achieve promising results, as discussed in the Chapter 6 Results and surpass traditional natural language processing methods, they have several disadvantages and limitations. This is particularly true when we focus on one of the main tasks of our work - ontology building. Large language models have demonstrated exceptional skill in acquiring patterns and representations from extensive text corpora. However, their effectiveness in addressing more complex tasks that require planning several steps and actions is limited. This is particularly evident when the required actions involve mathematical or physical reasoning [39], adherence to syntactic rules during source code generation, or the execution of complex data manipulation operations such as grouping, sorting, or other analytical transformations.

Research has shown significant improvement through the integration of the multimodality concept, which, besides the textual corpus, also incorporates diverse sources such as images and audio when training the models [15]. However, multimodality alone is insufficient for tasks such as arithmetic, geometry, chemistry, and mathematical equations. These tasks often demand specific reasoning abilities or domain knowledge that cannot be acquired by merely processing multiple data types. Research has shown that orchestrating actions using a predefined set of tools becomes a way to achieve significantly better results. Tools are critical components that enhance the capabilities of Large Language Models (LLMs). They are executable units with defined functions that allow LLMs to perform various tasks. By utilizing different tools, LLMs can devise a strategy to accomplish a broad array of objectives. For instance, we can develop an interface for an LLM agent to access tools capable of conducting activities beyond text generation, such as performing web searches, executing code, searching databases, or conducting mathematical calculations.

In today's era, the use of tools is the main driving force behind modern applications leveraging LLMs. The range of applications is vast, from customer support chatbots to complex financial advisors or psychological assistants for mental health [29]. One of the leading frameworks implementing the most empirically validated techniques and strategies is LangChain.

LangChain is an extensive library specifically designed to support the development of applications utilizing Large Language Models (LLMs). It facilitates the creation of sophisticated interaction workflows by integrating various components from multiple modules. In LangChain, an agent is constructed by combining tools and memory. The core principle of agents in LangChain involves using an LLM to determine an appropriate sequence of actions. Based on user input, the agent decides which tools, if any, to activate. Essentially, an agent in LangChain is a combination of tools and memory. After executing an action, the agent updates its memory, thereby maintaining the context of the interaction. This method allows LangChain to handle more complex and structured interactions, ensuring continuity and context across different prompts and responses.

In addition to the design of the tools themselves, well-known algorithms such as divide and conquer for decomposable tasks are commonly integrated. In particular, task decomposition results in a significant increase in the solve rate for more complex types of tasks [27]. LangChain is not the only framework for working with LLM agents; implementations of the Divide and Conquer or Additional Opinions algorithms can also be found in other popular frameworks. For example, the framework Auto-GPT [40] has proven to be a highly competitive contender with very effective performance in handling sophisticated and multifaceted tasks.

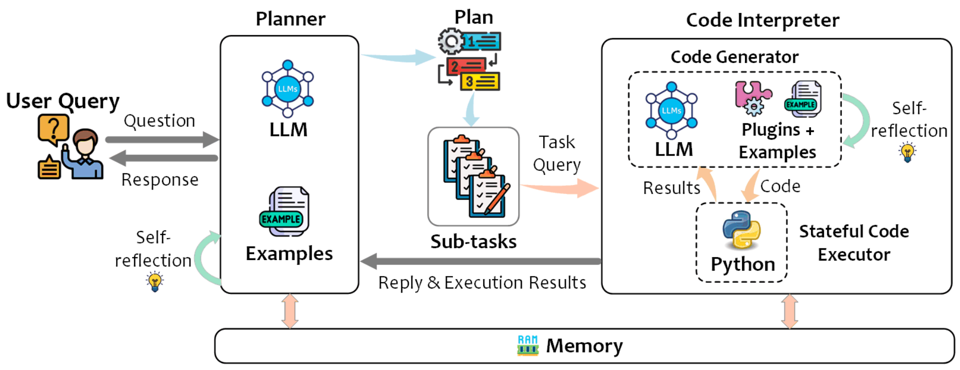


Figure 2.8: The architecture of TaskWeaver. This figure illustrates the main components and workflow of the TaskWeaver system, highlighting the interactions between various modules. Source: [16].

For the purpose of designing algorithms to extract terms and definitions and to conduct related experiments, we used a simpler framework similar to TaskWeaver (Figure 2.8). This framework is known as Semantic Kernel. Semantic Kernel is an open-source software development kit (SDK) designed to integrate large language models (LLMs) such as OpenAI, Azure OpenAI, and Hugging Face with traditional programming languages such as C#, Python, and Java. This framework enables the creation of advanced solutions for a variety of domains, including natural language processing, decision making, and information retrieval.

One of the main advantages of Semantic Kernel is its ability to automatically orchestrate plugins using LLMs. The framework allows to define and chain plugins that can be used to solve complex tasks. In addition, Semantic Kernel provides robust scheduling features that allow AI models to create and execute plans based on a user's specified goals. This approach facilitates business process automation and increases AI agent productivity by invoking existing code [18][17]. Developed by Microsoft, the Semantic Kernel offers robustness and reliability, ensuring high-quality performance and production level guarantees in various applications.

The fundamental concept of the framework is the definition of constructs that subsequently allow system designers to easily define a set of available actions using function decorators and class descriptors — semantic kernel functions and plugins. Behind the scenes, the developed tooling is transformed into the actual prompts selected by the planners. A planner is considered a prompt that implements one of the versions of state-of-the-art ReAct patterns. The distinction between planners most often lies in the format used for the final plan and its steps (XML, handlebars, etc.) and in the integration of new tips and tricks for using LLMs for planning. In the case of the Semantic Kernel, a plugin can be any snippet of code (e.g., MathPlugin), tooling that allows additional interaction with the user (e.g., UserInteractionPlugin), a database access interface, or additional LLM native plugins. One of the advantages is the capability of parallel execution of subtasks and supplementary tasks with additional LLM queries.

In the following section, we would like to present specific implementations of tested planners for our task. It should be noted that the ideas of the individual approaches demonstrated on the specific implementation in Semantic Kernel are, however, implementation-independent and reusable anywhere.

2.5.1 Action Planner

The action planner is specifically designed to orchestrate complex systems by selecting the most appropriate plugin for a given user intent. It works by first identifying the user's goal and then searching for the available plugins to find the most suitable one that can fulfill that goal with a single action. This approach ensures the efficiency and speed of the scheduler and is particularly useful in situations where quick decision making is critical and actions do not require sequential or multi-step reasoning. The scheduler uses LLMs to evaluate and select the plugin and sets the necessary parameters for execution. Once a suitable plugin is identified, the action planner generates a plan and immediately proceeds to async execute, thus completing the action without further iterations or feedback mechanisms. This makes Action Planner an excellent choice for direct, one-step actions where simplicity and speed are crucial.

2.5.2 Sequence Planner

The Sequence Planner, although no longer supported in favor of more flexible systems like the Handlebars Planner, was initially designed to facilitate the orderly execution of tasks by transferring outputs sequentially from one step to the next. This planner was particularly useful in environments requiring a linear progression of actions, such as data processing pipelines or automated task sequences in software development operations. By ensuring that each step was completed before the next began, the Sequence Planner helped maintain the integrity and dependency of complex task sequences, providing a structured approach to automation.

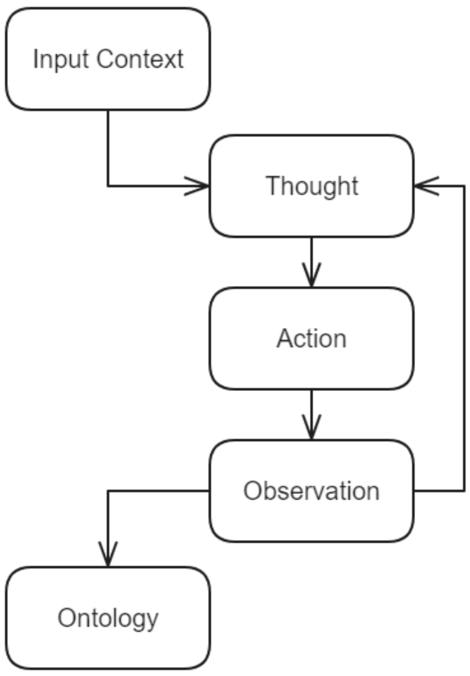


Figure 2.9: Simplified Stepwise planner ReAct pattern.

2.5.3 Stepwise Planner

The Stepwise Planner in Semantic Kernel emphasizes a well known, step-by-step approach to problem solving, making it ideal for complex scenarios involving multiple, interdependent processes. This planner employs a neuro-symbolic architecture, allowing for the dynamic selection and integration of the most appropriate plugins at each phase of the task. Such capability is vital in adaptive systems where decision paths may vary based on evolving data inputs or user interactions. The Stepwise Planner is particularly useful at managing tasks that require continual learning and adjustment, thereby optimizing the effectiveness of each step based on the outcomes of previous actions. This can lead to increased processing times and higher computational costs, particularly in scenarios with numerous or particularly complex steps.

Chapter 3

General Architecture

This chapter outlines the architecture and workflow required to achieve the objectives of the project. Each step in the process is detailed, including inputs, outputs, and descriptions of each component involved. The architecture is designed to ensure a systematic and efficient approach to ontology creation from legislative documents.

3.1 Overview

The architecture for extracting definitions from legislative documents and constructing an ontology consists of several key phases. Each phase is designed to ensure accuracy, efficiency, and coherence in the ontology development process and might depend on the results of the previous steps. The high-level steps include:

1. **Text Preprocessing**
2. **Term Extraction**
3. **Definition Extraction**
4. **Ontology Engineering and Validation**

3.2 Text Preprocessing

Inputs:

- PDF or DOCX file
- Manual text insertion

Outputs:

- Cleaned and formatted text ready for processing

Description: Text preprocessing is the foundational step in the workflow. It involves uploading the legislative document in either PDF or DOCX format or manually inserting the text. The system processes the text to remove special characters, hyperlinks, and other non-content elements using regular expressions and text processing libraries. This ensures the text is in a readable format, free from noise, and ready for further analysis. This step includes applying token limits to handle extensive texts effectively.

3.3 Term Extraction

Inputs:

- Preprocessed text
- Existing ontology terms (optional)

Outputs:

- List of candidate terms

Description: Term extraction involves identifying potential terms that could be included in the ontology. This process employs a combination of in-context learning (ICL) and few-shot examples to extract terms accurately. The input is the cleaned text, and the output is a list of terms that are forwarded for domain expert approval or correction.

3.4 Definition Extraction

Inputs:

- List of approved terms
- Legislative context

Outputs:

- Definitions associated with each term

Description: Definition extraction identifies and extracts definitions for the previously extracted terms. The process involves generating definitions using ICL and filtering out any terms not explicitly defined in the text. This step ensures that each term in the ontology is accompanied by a precise and contextually accurate definition. To enhance accuracy, the system incorporates pre-filtering to eliminate terms unlikely to be defined in the text and uses an index of approved formal definitions from previous extractions as reference points.

3.5 Ontology Engineering and Validation

Inputs:

- List of approved terms with their definitions

Outputs:

- Turtle/RDFXML/N-Triples source code
- Graphical preview

Description: The final phase involves consolidating all extracted terms and definitions into a coherent ontology. Each step includes manual validation by domain experts to ensure factual accuracy and correctness.

Chapter 4

Experiments

This chapter presents the experimental setup, methodologies, and results of various approaches undertaken to achieve the specified objectives. Each section presents specific partial experiments, highlighting the effectiveness and limitations of different techniques, and provides insights into their practical applications.

4.1 LLMs as POS Taggers

In our work we designed and benchmarked a way of POS tagging utilizing in-context learning of Large Language Models (LLMs) such as GPT-4. This tagger, bypassing traditional supervised learning, leverages the model's extensive pre-training on diverse linguistic data. The method is based on presenting the LLM with either preselected or dynamically chosen input-output pairs - a combination of partially tagged and untagged text. This design leverages the LLM's ability to infer and classify words into POS classes by recognizing patterns from the provided examples and applying these patterns to the untagged words. This supplementary task has been evaluated only on a small dataset and in any way is not intended as a replacement for already mentioned state-of-the-art POS Taggers. We evaluated this approach on the same dataset of 18 sections from construction law as in section 4.3. Since this experiment achieved similar accuracy to the Morphodita mentioned above - except for extreme cases, we did not pursue similar experiments due to increased response time and to save resources.

Utilization of pure Part-Of-Speech (POS) tagging for Term Candidate Extraction is fundamentally insufficient because POS taggers are primarily designed to categorize words based on their syntactic function (e.g., noun, verb, adjective), therefore they cannot capture the specific terminologies and phrases that do not conform to standard POS tagging categories. Challenges such as the Czech language's free-word order might further reduce the effectiveness of pure Part-Of-Speech (POS) tagging for Term Candidate Extraction.

4.2 Term Candidates Lemmatization

Another subtask of our work was to propose a suitable approach that would find for each term its lemmatized form, which we could use as an identifier (prefLabel or class name). The Czech language has many forms of word inflection, and therefore this situation occurs very often, for example, instead of *hladinou záplavy* the label of the term should be *hladina záplavy*. This could easily be done using the proposed Lemmatizers, but our terms, as shown, can be multi-word (phrases). However, the Lemmatizer considers individual words as separate entities, see Table 4.1.

Original	**MorphoDiTa**	**Stanza**	**Few Shot ICL**
areálem	areál	areál	areál
budovou	budova	budova	budova
bytem	byte	byt	byt
hladinou záplavy	hladina záplava	hladina záplava	hladina záplavy
hranicí	hranice	hranice	hranice
hrubou podlažní plochou	hrubý podlažní plocha	hrubý podlažní plocha	hrubá podlažní plocha
charakterem území	charakter území	charakter území	charakter území

Table 4.1: A selection of examples of term lemmatization by MorphoDiTa, Stanza, and Few Shot ICL.

4.3 Term Candidates Extraction Results

In the evaluation part of the proposed sub-components for term extraction, we used a part of the Czech Construction Law. More specifically first 18§ of

TP	FP	FN	Precision	Recall	F1 Score	Accuracy
32	75	10	0.30	0.76	0.43	0.76 (32/42)

Table 4.2: RAG Term Extraction - benchmarked on 18§ from *Zákon o územním plánování a stavebním řádu (stavební zákon)*

Zákon č. 183/2006 Sb. Mainly because this law has already been processed in the existing system, the terms and their definition extracted and revised by a domain expert. This fact allowed us to directly evaluate the proposed approach. Due to the time complexity, we limited ourselves to the part of the law containing the most terms and definitions. Specifically, the basic metrics can be seen in the Table 4.2. It should be mentioned that we compare the extracted terms with the existing ones using a Hamming distance equal to 0. Another necessary remark is the dependency on the configuration of the LLM queries, specifically the guarantee of the deterministic nature of the results. However, in the case of terms that we consider misclassified, quite often the term is very close to the correct one. In some cases, however, the hallucinations of the language model are apparent at first glance - for example, the incomplete term "*územní op...*". A large proportion of false positive cases would need to be verified by a domain expert, but for some, it is clear that it is in fact not a legal term (e.g. "*popřípadě republikového významu*")

4.4 Definition Candidate with Zero Shot RAG

If the domain expert accepts some of our term candidates, we can reuse the same approach to find their potential definitions. This is done again by RAG. As a result, we need to ask the domain expert to accept/correct the definition candidate.

4.5 Definition Candidate Extraction Results

Further experiments of the proposed approaches concern the extraction of potential definitions from already found and approved terms. A significant difference between the process of extracting term candidates and definition candidates lies in the very nature of the extracted entity. In the case of a term, it is a lemmatized word or phrase. In the case of a definition, however, it is very often a coherent piece of text that must directly reflect the legal text. It is the very semantic exactness and correlation to the original text that is one of the biggest challenges of this task and the most important metric for comparing different approaches. Our proposed approach allows extracting a definition either as part of the original text - just marking the area in which the definition is most likely to be found, in which case accuracy is guaranteed at the cost of the necessary subsequent reformulation by a domain expert. The second tested alternative is to use the full potential of RAG in combination with ICL and leave the formulation of the definition in its complete and closed form to the LLM, see Appendix A.3.

4.6 Initial Ontology

As a result of the previous steps, we have approved terms and their definitions, in addition, we also keep information about the source file (part of the law). The next step is the integration of the new terms into the ontology and their subsequent linking. Since at this point, the ontology is empty, we do not have to check whether the term or similar term is already in the ontology. We only need to insert it according to our schema. This includes creating appropriate persistent identifiers, one of which is the URI of the concept itself. Since our work builds on an existing system, we follow the already established conventions, see the appendix A.2.

4.7 Ontology Extraction Results

Finally, we have done experiments extracting the atomic ontologies themselves. It should be noted that the input of this component can be data from the already mentioned parts of the extraction pipeline, or data from an existing system. The evaluation of the results in this case is not as straightforward as in the previous experiments. Since in the current situation we cannot compare the quality and complexity of the extracted and built ontology with an existing one, we have no choice but to verify the correctness by judgment. The following process of creating a complex final ontology is linking the extracted subsets of Atomic Ontologies. This is the goal of our work in the near future.

4.8 Code Generation

Another approach on how to automatically generate ontologies is direct source code generation. This approach is based on the use of language models to generate code based on its specification and description in natural language query. The advantage of this approach is undoubtedly the speed and efficiency of the inference compared to the human resource requirements. Recently, this has been a very popular topic, even though the quality of the generated code can be very inconsistent and often requires revision and manual correction by the end user [12]. Another big concern in generating source code is security [38] [23], which is out of the scope of our work.

We decided to explore this area and conduct experiments with direct source code generation of atomic ontologies in N-Triples, RDF/XML and Turtle formats. These three formats are among the most popular ones [31] and each of them has different characteristics that have an impact on the token inference. Let us first describe each format to give an overview of their advantages and disadvantages. For each of them, we will also give an intuition for why the format should be more suitable for the language model or vice versa.

N-Triples is one of the most basic and simplest formats for serializing RDF data. Each statement is on a single line and consists of a subject-predicate-object triple terminated by a period. These characteristics (assumptions) and standardizations are what make it suitable for machine processing - it is very easy to parse. However, its excessive simplicity is also a major drawback, as it is less human-readable, especially when it contains more complex data structures. For the purpose of direct ontology snippet generation, this format is particularly suitable because of its syntactic simplicity, which minimizes generation errors. Another advantage is the ease of incrementation, due to the complete independence of the rows, which also allows incremental testing.

RDF/XML, on the other hand, is a much more syntax-rich format, allowing the representation of more complex data structures, which is what we need in the case of more complex ontologies. A major advantage is the popularity of the XML format, which often facilitates rapid integration into existing systems and libraries. However, the complexity of the syntax significantly complicates processing and, from the perspective of generation using language models, the actual inference time and the number of tokens consumed.

Turtle is one of the many formats that are very readable and easily modifiable by humans. Another advantage based on good readability is ease of maintenance and sustainability. The conciseness and clarity of the syntax, however, require the use of special characters and forms of truncation (and prefixes) which result in significantly higher error rates when using language models [8].

4.8.1 Format Comparison Conclusions

When we look at how different formats perform in generating ontologies using language models, we see differences in error rates and how many tokens they use. The Turtle format, which is easy to read and modify, makes the most mistakes during automatic generation because it uses special characters and shortcuts that are easy to get wrong. On the other hand, N-Triples, although very straightforward because it uses one line for each data point, ends up using more tokens, which isn't efficient for storing complex information. However, its simple structure reduces the chance of making errors. RDF/XML is somewhere in between; it can handle complex information well because it uses XML, but this also makes it harder to process, affecting both the number of tokens used and how often errors occur.

4.8.2 Naive Code Generation Conclusion

Our initial experiments with direct code generation for ontologies demonstrated a very high error rate, with mistakes occurring in more than 95% of cases - model *gpt3.5-turbo*, turtle format with missing semicolons as the most common issue. This level of inaccuracy indicates that this method is not suitable for our specific task of creating ontologies. The high frequency of errors made it clear that continuing to test this approach would be inefficient, particularly in terms of using up computational resources - inference tokens. Before transitioning to a completely different approach, we modified the initial code generation component to include a feedback loop from subsequent parsers. As detailed in the next section 4.9, this adjustment effectively reduced syntax errors. However, it came at the cost of a significant increase in the number of tokens consumed and the response time.

4.9 Code Generation With Feedback Loop

From the experimental results of the proposed algorithm for extracting terms and definitions in the previous chapter - code generation, it is evident that the use of the LLMs for direct generation of the source code of the ontology itself in any of the above mentioned formats from the input legal document is inappropriate due to the very high error rate - common syntax mistakes, not just semantically incorrect ontology.

Therefore, in further experiments, we modified the previous design and introduced an error feedback loop. This is the idea of integrating an already existing component in our system validating the syntax of the generated code into the ontology extraction/generation process itself, see Figure 4.1. The first step is identical to the previous proposal - ontology generation from input text using the principle of in-context learning and few-shot prompting. The second step is validation of the generated code.

Validation of source code syntax in any format is performed using existing parsers implemented in commonly used libraries for RDF/XML or turtle data. Specifically, we used the rdflib library for N-Triples and Turtle, and the owlready2 library for validating RDF/XML data. Both of these libraries support catching error messages. Error messages, in addition to the source code itself, most often contain a brief description of the syntax error and a reference to a specific line - error line.

It is the description of the error message and the reference to a specific line in the input file that we used as feedback for the next generation cycles (see Figure 4.1). This change led to a very significant improvement in the quality of the resulting extraction (ontology source code), reducing the error rate from approximately more than 95% to range between 25% to 30%. However, the remaining cases became stuck in a loop, presenting an unresolved issue. This was tested on atomic ontologies no more than 1000 tokens long.

However, it is necessary to point out the disadvantages of this approach. One of the main drawbacks is the significant increase in total response time - each error correction step requires another LLM call, leading to chained calls due to the high error rate of the trivial approach. In our experiments, this resulted in a several-fold increase in response time. Given the chosen LLM models, we also noted an increase in the number of inferred tokens, which in turn led to higher spending.

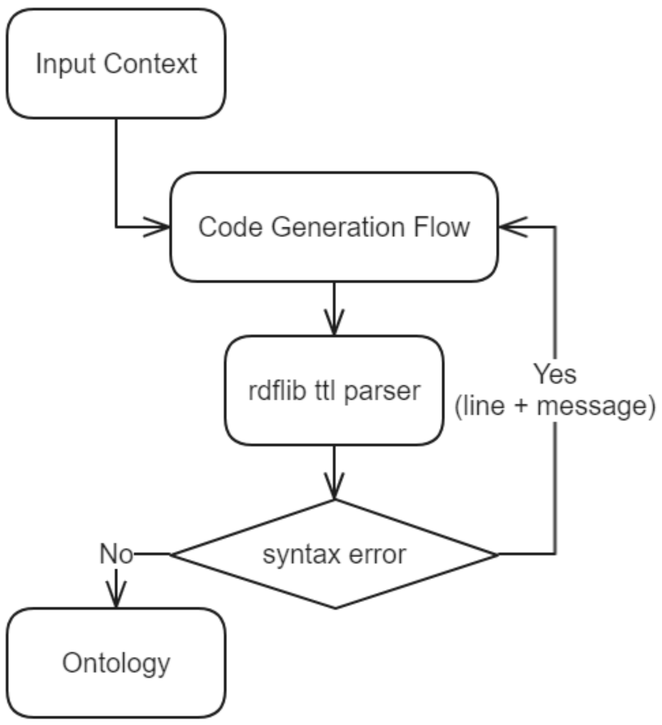


Figure 4.1: Diagram demonstrating implemented code generation with error feedback loop.

In the next steps of our system design, given these results and considering all the drawbacks, we backed away from this approach and proposed an alternative extraction algorithm. Instead, we have shifted our focus to an orchestration approach, where the decision-making process is handled by the model, and the actual implementation is executed using existing Python libraries. This method allows for more controlled and error-free ontology creation.

Chapter 5

Proposed Solution

In this chapter, our goal is to consolidate knowledge about the state-of-the-art methods and models, and most importantly, the results of previous experiments discussed in Chapter 4. The primary objectives are term extraction, definition extraction, and subsequent ontology engineering. Additionally, we incorporated SubClass extraction and optional ObjectProperty extraction processes to facilitate the construction of more complex ontologies. We propose two suitable approaches: OntoBuilder and OntoAssistant. To ensure the factual accuracy of the entities integrated into the ontology, we mandate domain expert review and potential corrective actions after each step.

5.1 OntoBuilder

The first approach involves OntoBuilder - a collaborative tool wrapped into a web application. It contains separate flows designed for the dynamic extraction, approval, and subsequent integration of entities into the developing ontology.

5.1.1 Overview

The process is divided into five distinct phases: Upload/Preprocessing, Term Extraction, Definition Extraction, SubClass Extraction, and, optionally, Ob-

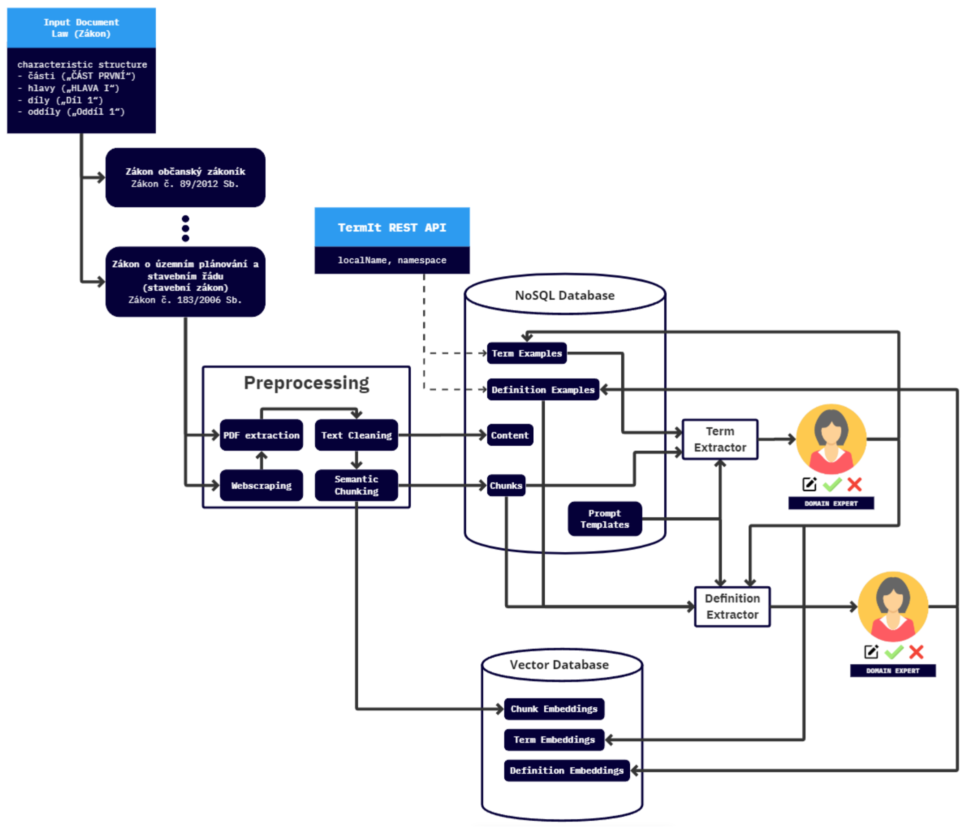


Figure 5.1: Data flow diagram of the proposed application. The diagram outlines the main extraction components and their interactions, illustrating the flow of data through various stages of processing within the system.

jectProperty Extraction. This sequence culminates in the final revision of the ontology and a graphical preview.

5.1.2 Text Preprocessing

In the first step of the proposed algorithm, it is necessary to specify the input text region from which we want to extract Term Candidates. The proposed system allows two approaches

- uploading a PDF/DOCX file
- manual text insertion

Manual text insertion is a simpler option, but it does not allow convenient work with larger documents. However, an advantage of this option is the unnecessity of text preprocessing, because we assume that the inserted text is already in readable form (no special characters, no headers/footers, correct encoding, etc.).

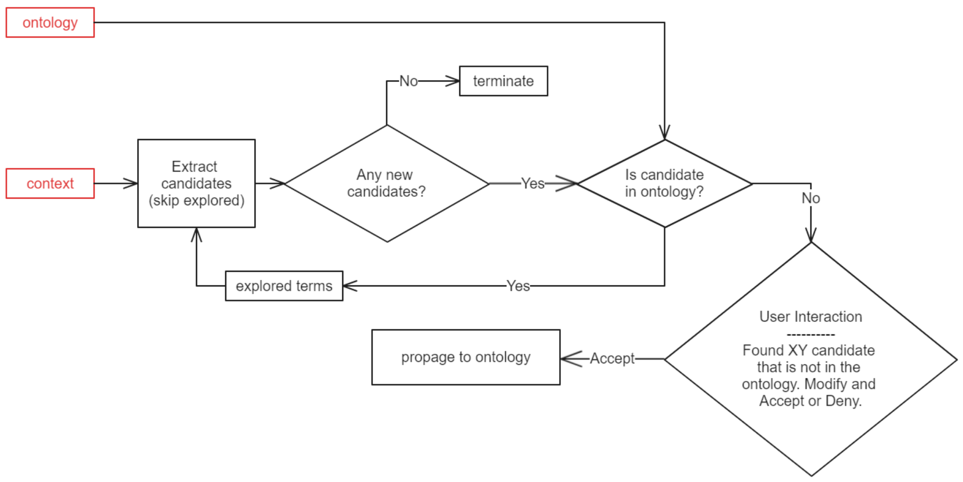
The second option is to upload a PDF or DOCX file. After successfully uploading the file to the server, the system reads the content of the document and saves it in a persistent text format. The basic steps of rule-based text preprocessing are then applied using regular expressions. Most files contain additional special characters, hypertext links, image objects, or text unrelated to the content itself (page footer in small font). For this purpose, we use available libraries that allow us to retrieve, in addition to the content itself, metadata about font size, tags and other features.

Due to the models used, we also introduced an upper limit on the number of tokens to avoid input contexts that are too long. A token is a part of a word. Specifically, in our work we use the encoder *cl100k_base*, which is used by the chosen models. For the purpose of testing of the proposed components, we chose a limit of 2000 tokens. The amount of text that can fit into 2000 tokens varies greatly based on the characteristics of the text itself. In the case of the Czech language, the number of tokens per word is 2.11 times higher than the English [24]. In the case of the test scenario on the construction law, this is approximately 4000 characters or 2 standard pages.

5.1.3 Existing Ontology

One of the main advantages of working with open data is its ease of integration and extensibility. To preserve this advantage within the proposed system, we allow the domain expert to upload his own ontology. Specifically, an ontology containing an already non-trivial set of pre-validated terms represented as OWL Classes, optionally with definitions. In the current version of the proposed solution, our focus is limited to utilizing pre-existing terms. However, future iterations will extend reusability to the text of the definitions themselves. This extension will involve comparing the definitions using the cosine distance of embeddings against generated definitions. For terms, this process is relatively straightforward due to the establishment of a normalized format for the class identifiers representing each term.

The set of terms already existing in the ontology is embedded in the term extraction flow by injecting them into the prompts themselves. At the same time, the system part of the prompts is dynamically modified so that existing


ontology
context
Extract candidates (skip explored)
Any new candidates?
No
terminate
Yes
Is candidate in ontology?
No
Yes
explored terms
User Interaction

Found XY candidate that is not in the ontology. Modify and Accept or Deny.
Accept
propage to ontology

Figure 5.2: Flowchart draft of the ontology enrichment process. The diagram illustrates the steps for extracting and validating candidate terms against an existing ontology, including user interactions for term validation and propagation of accepted terms into the ontology.

terms are ignored in the first extraction step.

An alternative approach involves extracting all terms initially and then performing filtering at the output stage of the term extraction, prior to approval and propagation to the ontology. However, this method is less efficient as it results in unnecessary inference of tokens, thereby increasing costs.

5.1.4 Term Extraction

Based on experiments with classical methods for extracting entities declared as candidate terms and evaluation of state-of-the-art methods for entity extraction, we decided to use a modified ICL approach. The input of the pipeline for term candidate extraction is a preprocessed input chunk that has been revised and approved by the domain expert in the previous step. We also assume that it is part of a legal document containing legal terms. In the case of using the Ontology Enrichment option, the input is also a list of existing terms. The output of the Term Extraction flow is a list of candidates that is forwarded for approval or correction.

The main component of the Term extraction pipeline is the extraction prompt itself, which consists of several parts:

Task Description

The key to the success of any interaction with a language model is an accurate and structured task description. In our case the main objective is to identify concepts representing a Legal Terms and return them as a list of candidates. It is worth to mention that we observed significant increase in quality of the responses with more specific description - e.g. input is section of Czech Legal Document and extracted terms will be represented as owl2 classes in next steps.

Another tested and common practice is the detailed specification of instructions, guidelines or step by steps, which is represented by a list of actions that should be followed. Based on experiments, we have finalized and specified the following instruction categories Class Identification, Nomenclature, Precision, Exclusivity, Structure, Handle No Candidates, Exclusion. The full prompt is given in the attached materials.

Few Shot Examples

As already mentioned in the theoretical section introducing in context learning, research has shown that examples, including the solution, have a significant impact on the quality of the output. For the purpose of term extraction, one sample example is a pair of input text and a list of domain expert approved terms. We store the examples in the Azure AI Search, see section 6.1, index during the use of the application. In addition to the mentioned pair, the content of a particular entry is the embedding (vector field) of the input text and the identifier of the user who approved the term.

Subsequent retrieval is based on the similarity of the embedding of the input text to previously embedded contexts. We chose the Hierarchical Navigable Small World (HNSW) search algorithm and the cosine distance similarity metric. This is the most common combination and chosen as the default in the framework we used. Due to the very small set of matched entities, we did not observe any differences when using Exhaustive K-Nearest Neighbors. It should be mentioned that the main objective of our work was not to compare and benchmark the different retrieval methods or embedding comparison metrics. A set of mostly 3 to 5 sample examples that were most similar to the input is then formatted and dynamically embedded in the system part of the prompt.

This step contributed significantly to the improvement of the extraction quality and robustness in our case. In addition to the quality of the extracted data itself, this approach allows the user to dynamically adjust the level of granularity of the extracted terms. This aspect is particularly important when dealing with complex legal texts, where it may be necessary to adjust the level of granularity of the extracted information according to the specific needs of the user. Consider the following legal text as an example: *(1) V tomto zákoně se rozumí: m) veřejnou infrastrukturou pozemky, stavby, zařízení, a to: 1. dopravní infrastruktura, například stavby pozemních komunikací, drah, vodních cest, letišť a s nimi souvisejících zařízení;* In this context, the LLM can identify the following terms as relevant legal terms: *veřejná infrastruktura* (public infrastructure) and *dopravní infrastruktura* (transport infrastructure). This selection reflects the higher level of granularity that a user might prefer if they have requested more general categories of terms in the past. However, the flexibility and dynamic granularity settings also allow the extraction of more detailed terms such as *pozemek*, *stavba*, *zařízení*, *dráha*, *vodní cesta*, *letiště*, etc. This approach, based on few-shot prompting, allows the model to adapt to user preferences based on previous interactions and context.

Existing Terms

The last part of the system prompt is the existing terms contained in the uploaded ontology. At the same time, the system dynamically modifies the system prompt instructions to ignore the already existing most similar terms.

Infinitive Form

The next part of this pipeline takes care of the subsequent formatting. Again, we used ICL approach. But this time the main objective is completely different from the previous step. The input is a list of extracted candidates, for which we cannot guarantee any formatting style, infinitive or lemmatized form, etc. The system prompt contains instructions and information about the input terms itself, in addition to static examples of lemmatization, conversion to base form, and formatting. The output is a list of terms in the form corresponding to the identifiers of specific concepts - e.g. *wastewater-treatment-plant.*

The entire extraction process is wrapped into a single instance of Prompt flow, allowing for isolated deployment. A major advantage of encapsulating this component in a separate endpoint is the integration into other systems and endpoints.

5.1.5 Definition Extraction

Similar to the term extraction, we proposed the following approach to extract definitions based on previous experiments. We tried both the approach of labelling parts of the original text and direct definition formulation using LLM. Due to the very high consumption of inference tokens and the similar quality of the output, we opted for the direct definition formulation approach.

Task Description

This means that this pipeline again takes as input part of the text from the legislative document and a set of existing approved terms. The main goal is to recognize if any of the terms are defined in the context and formulate the appropriate definition. The output is a set, it can be empty, which corresponds to a scenario where the context does not contain an unambiguous legal definition of either of the terms, of term-definition pairs.

Pre-filtering

The initial version of the proposed extraction algorithm had one major drawback. In the case where the user required working at a very low level of term extraction granularity - the input term set contained many terms that were not defined in the text - hallucinations and unsubstantiated definitions occurred very often. In order to eliminate this problem as much as possible, we implemented a component before the actual definition extraction process that uses LLM to filter the terms that are not defined in the text.

The task is again solved using ICL, where the main goal is to identify in the input list of terms those that are not directly defined in a particular context and filter them out for the purpose of definition extraction.

Few Shot Examples

In order to improve the quality of the definition extraction results, we created an index storing the approved formal definitions from previous extraction iterations. This time the structure of the index is different, but the principle is the same. Again, the input context is compared with embeddings of contexts already stored in the index, and the output of the query is a set of terms including their formal definitions. This means that a set of terms extracted from the most semantically similar region of the text is always dynamically inserted into the prompt. At the same time, the examples, in addition to providing information about what the user has previously considered to be a definition and what not, contain useful information about the wording and stylistics of the definition itself.

At the end of the definition extraction is postprocessing, which takes care of checking and possibly removing model hallucinations, for example in term identifiers, or treating error outputs.

5.1.6 SubClass Extraction

Since the approach of the OntoBuilder algorithm is directly tied to the design and definition of the extraction steps and the revision interaction with the domain expert, it is not entirely straightforward to integrate the full expressiveness of OWL. In other words, if we wanted to extract from the input text, in addition to terms represented as ontology classes and definitions as ontology class labels, relations between single entities, we are not allowed to do so. Therefore, we decided to enrich the ontology creation steps at least with the extraction of parent-child relationships and experimentally in the 5.1.7 section also with Object Properties.

The task of SubClass relation extraction again requires a legislative context and a set of found terms as input. Similar to the previous steps, we empirically developed and modified system instructions to achieve the desired objective. In particular, we focused on introducing the context of the task performed, describing and specifying the subClassOf relations - transitivity, directionality, equivalence. As in the case of definitions, it was necessary to modify the task description for the scenario where there is no parent-child relationship between the entities in the input text.

5.1.7 Object Property Extraction

An additional component is to find candidates for Object Properties. The input is a set of approved candidates for terms and legislative context. The main goal of this flow is to find potential candidates for Object Properties. For this part too, we applied modified ICL methods and customized the system prompt as follows. The main goal is to find the set of entities that we identify with ObjectPropertyName. In addition to the identifier, each entity must contain a Domain and Range class, which are references to existing classes in the ontology representing specific terms. Next, a short label explaining the nature of the relationship. The last part is optional InversePropertyName if the relationship is bidirectional.

5.2 OntoAssistant

The OntoAssistant represents a collaborative, chat-based approach to the extraction of terms and definitions, and their subsequent incorporation into ontologies. This method was developed as a fully alternative approach to OntoBuilder, primarily due to the latter's limitations, including its inability to integrate user feedback effectively. The deterministic nature and occasional inconsistencies of OntoBuilder highlighted the need for an approach that could reiterate certain steps based on feedback from domain experts, with modified instructions and objectives. Unlike OntoBuilder, which follows a strictly defined scenario path, OntoAssistant allows for adjustments that are not generalizable within a standard architectural framework.

In the current version, we have designed two sets of tools. The first set enables the planner to communicate with the user and inquire about necessary details through the User Interaction Plugin. This plugin facilitates three functions: *Ask For Context*, *Ask For Confirmation Or Correction*, and *Ask To Continue*. The first function allows the planner to ask for the input text where terms and definitions are to be searched (currently not linked to an upload/storage component). This process can also accommodate specific user needs, such as identifying terms through their abbreviations instead of full phrases. The second function enables the correction and confirmation of candidate terms to be propagated into the ontology. This interactive step also allows for further specification of post-processing adjustments to the found terms. The *Ask To Continue* (see Figure 5.3) function is designed to facilitate continuous enrichment of the ontology, where a Yes/No response from the user decides whether to jump to the first step – querying for context specification or terminate.

The second set of tools is designed for manipulating entities within the ontology through the Ontology Plugin. This involves straightforward get, set, and delete operations mapped onto the syntax used by the owlready2

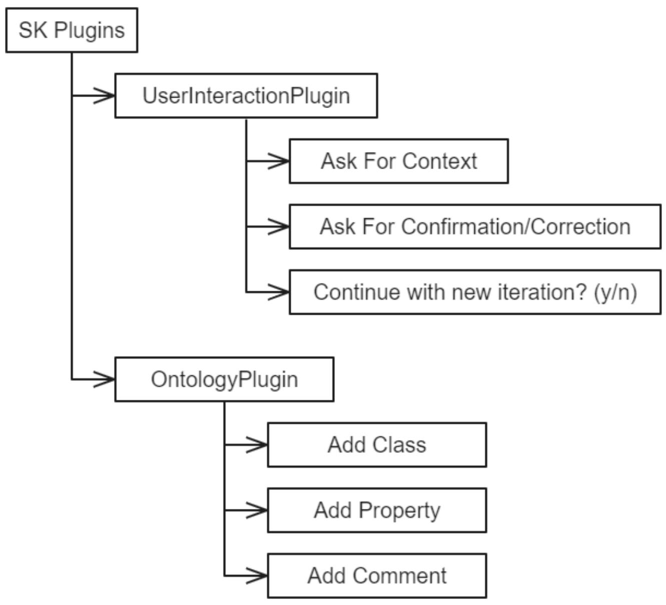


Figure 5.3: Partial hierarchy of designed tooling - semantic kernel plugins.

Python library. To reduce the planner's necessary steps and decrease response times, functions such as AddTerm and AddDefinition were introduced. These functions streamline the process by correcting identifier formats and managing issues related to the potential absence of non-existent objects.

Given that this component is still in the experimental stage of development, access to domain experts has not been enabled. Therefore, no partial observations or results are reported in subsequent chapters. Given that this is an auxiliary task designed as an alternative to established and tested methods and components, its experimental nature does not interfere with the achievement of the main objectives of this thesis.

Chapter 6

Results

Since automatic validation of the correctness and completeness of the created ontologies is almost impossible due to the open world assumption and the state of open data in the Czech legislation, we decided to directly contact domain experts who, in addition to their knowledge in the specific domain, have experience with open data. In order to run a series of manual test scenarios, we designed the architecture and integrated the OntoBuilder approach into the web application.

6.1 Web App for Domain Experts

Prompt flow - Extraction Components

The term, definition, subclass and object property extraction components are implemented and wrapped in separate Microsoft prompt flow instances. Prompt flow is an integrated suite of development tools aimed at optimizing the comprehensive development cycle of large language model (LLM)-based artificial intelligence applications. This suite covers the stages from ideation, prototyping, testing, and evaluation to production deployment and monitoring. By simplifying prompt engineering, Prompt flow facilitates the creation of LLM applications that meet production quality standards. The organization and connection of the individual steps by means of the Directed Acyclic Graph (DAG) allows you to work clearly and very efficiently with different variants of the solution. A great advantage is the possibility of simple encapsulation of the flow in a docker image or direct deployment in the form of an inference REST api endpoint. For testing purposes, we call individual flows directly from the backend of the web application.

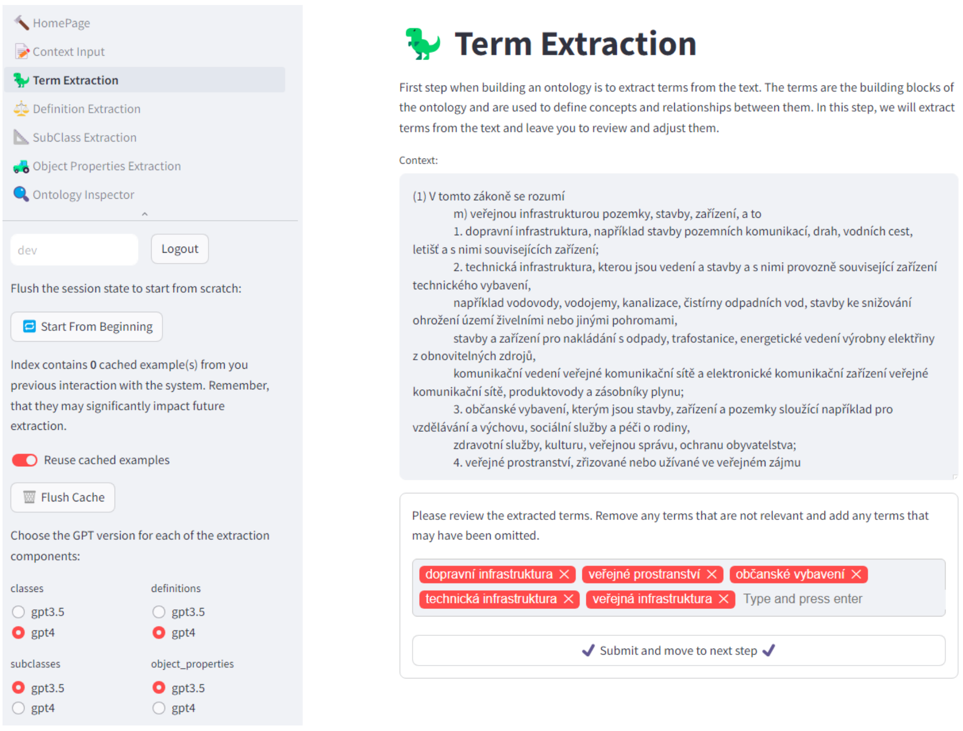
Streamlit - Web App

The web interface for our project was developed utilizing the open-source Streamlit framework. Streamlit is renowned for facilitating rapid development of simple web applications using Python. It is currently one of the most prominent tools available for creating and deploying web applications without requiring any expertise in backend or frontend technologies. For our purposes, we developed a multi-page application that guides a domain expert through a step-by-step process of extraction and ontology engineering. The application consists of the following pages:

- Home Page
- Context Input Page
- Term Extraction Page
- Definition Extraction Page
- SubClass Extraction Page
- ObjectProperty Extraction Page
- Ontology Preview Page

The Home Page provides the domain expert with essential information regarding the application's functionality and usage. The extraction process starts on the Context Input Page, where the user can either upload and preprocess a legal document or manually input text. Additionally, there is an option to upload an existing ontology to be enriched with new entities.

The subsequent extraction process is carried out through the aforementioned components. The interactive user interface is designed to facilitate easy revision and modification of the extraction results. It includes multi-choice forms, checkboxes, and text fields for correcting definitions, among other functionalities.



Flush the session state to start from scratch:

Index contains **0** cached example(s) from you previous interaction with the system. Remember, that they may significantly impact future extraction.

Reuse cached examples: on

Choose the GPT version for each of the extraction components:

classes: gpt3.5 [✓ gpt4]

definitions: gpt3.5 [✓ gpt4]

subclasses: [✓ gpt3.5] gpt4

object_properties: [✓ gpt3.5] gpt4

Term Extraction

First step when building an ontology is to extract terms from the text. The terms are the building blocks of the ontology and are used to define concepts and relationships between them. In this step, we will extract terms from the text and leave you to review and adjust them.

Context:

```
(1) V tomto zákoně se rozumí
        m) veřejnou infrastrukturou pozemky, stavby, zařízení, a to
        1. dopravní infrastruktura, například stavby pozemních komunikací, drah, vodních cest,
letišť a s nimi souvisejících zařízení;
        2. technická infrastruktura, kterou jsou vedení a stavby a s nimi provozně související zařízení
technického vybavení,
        například vodovody, vodojemy, kanalizace, čistírny odpadních vod, stavby ke snižování
ohrožení území živelními nebo jinými pohromami,
        stavby a zařízení pro nakládání s odpady, trafostanice, energetické vedení výrobny elektřiny
z obnovitelných zdrojů,
        komunikační vedení veřejné komunikační sítě a elektronické komunikační zařízení veřejné
komunikační sítě, produktovody a zásobníky plynu;
        3. občanské vybavení, kterým jsou stavby, zařízení a pozemky sloužící například pro
vzdělávání a výchovu, sociální služby a péči o rodiny,
        zdravotní služby, kulturu, veřejnou správu, ochranu obyvatelstva;
        4. veřejné prostranství, zřizované nebo užívané ve veřejném zájmu
```

Please review the extracted terms. Remove any terms that are not relevant and add any terms that may have been omitted.

dopravní infrastruktura ✕ | veřejné prostranství ✕ | občanské vybavení ✕ | technická infrastruktura ✕ | veřejná infrastruktura ✕ | Type and press enter

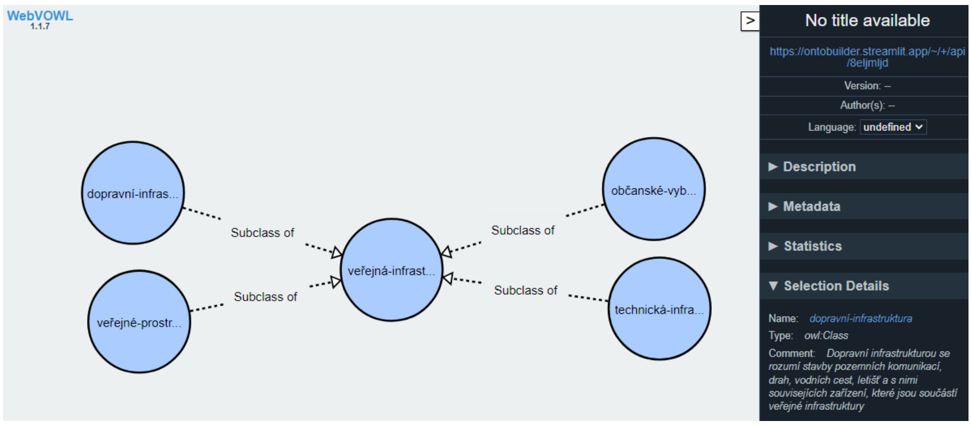
Figure 6.1: OntoBuilder Web Application - Term Extraction Page. Interface enabling domain experts to test and validate the proposed ontology extraction and engineering solution step-by-step.

In the final step, users are redirected to the Ontology Preview Page, where they are presented with a visual preview of the ontology generated using WebVOWL technology[1], see Figure 6.2. Users can also inspect the ontology source code in Turtle format. The final ontology can be downloaded, or users can return to the Context Input Page to start a new extraction iteration, thereby enriching the ontology with information from a new context.

Azure AI Search - Index & Retrieval

To efficiently and dynamically adapt the components for term extraction and definition extraction, we store user-approved outputs from previous application runs. For effective indexing and subsequent searching, employing techniques mentioned in subsection 5.1.4, we utilized services available on Microsoft Azure. Specifically, we used their Azure AI Search service, which allows for the storage of data structures and corresponding embeddings. The advantage of this service lies in its configurable search system; in our case, we utilize vector search based on the cosine distance of specific contexts.

[1] `https://service.tib.eu/webvowl/`



No title available

https://ontobuilder.streamlit.app/~/+/api/8eljmljd

Version: --

Author(s): --

Language: undefined

Name: dopravní-infrastruktura

Type: owl:Class

Comment: Dopravní infrastrukturou se rozumí stavby pozemních komunikací, drah, vodních cest, letišť a s nimi souvisejících zařízení, které jsou součástí veřejné infrastruktury

Figure 6.2: Result ontology preview with WebVOWL, *veřejná infrastruktura* (public infrastructure) as the parent class with subclasses: *dopravní infrastruktura* (transport infrastructure), *občanské vybavení* (civic amenities), *technická infrastruktura* (technical infrastructure), and *veřejné prostranství* (public spaces). The side panel provides detailed information about the selected node.

MongoDB - History & User Management

In addition to cloud services for indexing and subsequent searching, our application also uses one of the available NoSQL database offerings. Specifically, we use MongoDB to store the history of interactions with the application. Keeping records of each step is crucial for processing feedback. The recorded extraction results, manual adjustments, timestamps, and any error messages have enabled us to effectively fix issues and improve the quality of the results.

6.1.1 Token Usage and Inference Costs

The web application is hosted and publicly available using the free services of the Streamlit Community Cloud. Streamlit Cloud is an ideal choice due to its generous free resources, seamless setup of CI/CD pipelines, and native integration with the library itself. An alternative solution, which we have not utilized at this time, is the deployment of the application as an Azure Web Application service. Because, for our purposes, the hardware available in the free version is insufficient, and we would need to use higher paid tiers.

We host the mentioned Azure services within the available free tiers, resulting in zero hosting infrastructure costs. Undoubtedly, a significant advantage of the chosen type of infrastructure is the seamless upscaling capability if needed.

During the experiments, development, and subsequent testing of the proposed solution, we used OpenAI models *gpt-3.5-turbo* and *gpt-4-turbo*. Some simpler sub-components use the cheaper and faster model. On the other hand, for the main components, the choice is left to the user through settings in the web interface, with the default being *gpt-4-turbo*. The total consumption of chat completion and embedding tokens (model *text-embedding-ada-002*) was 4.9 million tokens, resulting in costs ranging from 25 to 30 USD.

6.2 Testing

In this section the goal is to present the design, steps, and results of manual testing by domain experts. As previously mentioned, due to the collaborative components and open solution assumption, we opted for evaluation by potential future users instead of partial and potentially incomplete automatic validation. The primary objective was to verify the functionality and usability of the collaborative extraction of terms and definitions through the OntoBuilder web application. For this purpose, we contacted domain experts from the Institute of Planning and Development of Prague (IPR Prague), who were provided with access and instructions for using our application. Additionally, we prepared an expected walkthrough of the application in the form of a sample testing scenario, which was designed to demonstrate the required outputs, metrics, and feedback for the testing.

The test scenario consists of parts focused on individual components presented in section 5.1. Primarily, it involves questions concerning residual or missing candidates, correctness of formulation, necessity of manual adjustments, factual accuracy, overall quality assessment of the extraction, and verbal feedback. For simplicity in testing and subsequent test evaluation, we defined quantitative metrics as categorical, allowing testers to choose from five values: none, few, moderate, many, numerous (e.g., *Were there any terms missing that were contained in the text?*). When evaluating the test results, these categorical values were mapped to a scale from 0 to 1 (worst 0, 0.25, 0.5, 0.75, 1 best). In addition to these metrics, we kept track of the overall rating and subjective impression of the quality of individual parts on a scale of 0 to 10, also remapped to 0 to 1. Lastly, we allowed testers to comment on specific parts of the system and verbally describe any deficiencies. In the final step of our testing process, we focused on the practical benefits of the system, specifically examining its effectiveness in facilitating workflow and speeding up the transformation of input text into an ontological structure, as compared to a purely manual approach.

Below we present the average results from a total of six testing scenarios done by two domain experts. A primary advantage of manual testing is that it does not require a ground truth for results. However, the drawback is its time-consuming nature. Experts need to be introduced to the application, familiarized with its controls and expected usage. "Additionally, experts must allocate time to understand the testing objectives and to proceed through the test scenarios. For these reasons, even a seemingly small number of scenarios took testers between one to two hours to complete.

Term Extraction

We assessed the quality of term extraction in our testing scenarios using the following specific questions:

- Were any terms present in the context omitted in the extraction?
- Were any terms extracted that were not present in the context?
- Were any of the extracted terms in an incorrect form?
- What was the proportion of correctly extracted terms without the need for manual adjustments?
- What is the overall quality rating of the Term Extraction?

All subsequent results were scaled to a range of 0% to 100%, where 0% always represents the worst quality, accuracy, or correctness of the output. On the other hand, 100% indicates perfection, reflecting the best possible outcome achieved. For instance, if a tester answered "none" to the question of whether any of the extracted terms were in an incorrect form, the output is 100%.

In the case of checking for missing/extra terms, testers more often encountered extra terms than missing ones, with specific instances recorded at 87.5% and 79.17%, respectively. They also noted that the form of the extracted terms was correct in all scenarios, including conversion to their base forms, achieving a 100% accuracy. In four out of five cases, the testers did not need to make manual adjustments to the outputs, declaring them correct and proceeding to the next steps. The average overall rating and subjective impression of the quality of the term extraction component from the input legislative text was 85%. The most common critique in the verbal feedback category related to the excessive candidates (false positives). Additionally, testers provided feedback on the application's usability, noting that the alphabetical ordering of candidates does not match the order in which they appear in the context, which slows down subsequent checking and correction.

Definition Extraction

The subsequent area of testing was focused on the evaluation of extracted definitions. We assessed the quality using the following specific questions:

- Were there any missing definitions that were actually contained in the context?
- Were there any additional definitions extracted that were not actually contained in the context?
- What proportion of extracted definitions are directly supported by the context?
- What proportion of definitions were factually correct without the need for manual corrections?
- What is the overall quality rating of the Definition Extraction?

The feedback on the definition extraction process was highly positive. Specifically, testers noted no missing or unnecessary definitions, achieving 100% accuracy in both categories. Minor hallucinations and inaccuracies were observed in only one testing scenario, affecting the response rate for the third question to 93.33%. Similarly, factual inaccuracies necessitated manual intervention in 3,33% of cases. Despite these minor issues, they did not influence the overall and subjective rating of quality for this component, as both testers across all scenarios provided the maximum possible rating of 100%. Verbal feedback underscored the system's effectiveness, noting, *It delivers very good results and significantly reduces manual labor required for definition extraction.*

SubClass Extraction

The final mandatory component for testing involved the extraction of parent-child relationships. Given that the testers were not required to ensure that the input text had to contain entities between which this relationship existed, we asked this question separately for each scenario. If testers responded negatively, indicating that no such entities existed within the context to define this relationship, they could skip testing this component. On the other hand, if they responded positively, they were further queried with these questions:

- What proportion of the existing SubClass relationships was accurately identified?
- What is the overall quality rating of the SubClass relationship extraction?

The testing results were mostly positive, with correct outputs reported at 88% and 86%, respectively. Testers commented on occasional missing instances of one of the descendants and also on additional relationships that could not be directly classified as parent-child relationships, noted *A pertinent hierarchical relationship was identified accurately, and an additional irrelevant relationship was proposed but subsequently rejected.*

6.2.1 Other

Finally, we asked domain experts for feedback regarding the speed of ontology creation processes with basic entity types, from downloading specific legislation to the final ontology version, when conducted manually versus using our methods. The primary objective of this inquiry was to determine whether the algorithms integrated into a collaborative user environment could save time for open data experts and enhance their workflow efficiency. In all tested scenarios, both testers reported that the OntoBuilder application enabled them to achieve the same results faster than they would manually.

Metric	**Time**
Fastest Time	4m 30s
Longest Time	13m 54s
Avg. Time	9m 19s

Table 6.1: Summary of Test Scenario Completion Time

Through the application's history tracking feature, we monitored the time it took to process text segments from documents (each containing up to 2000 tokens) throughout the step-by-step ontology development [2]. The process

[2]Testers chose not to engage with the optional task of extracting ObjectProperties.

varied in duration: the quickest completion took 4 minutes and 30 seconds, whereas the longest took 13 minutes and 54 seconds, with an average of 9 minutes and 19 seconds per scenario, see Table 6.1. Please note, these times exclude the periods spent getting familiar with the application, configuring its settings, and managing the contexts.

Chapter 7

Conclusion

In the introductory section of our work, we familiarized ourselves with semantic web technologies, including the most commonly used formats and techniques, with a particular focus on ontology creation. However, to begin working on term and definition extraction, we needed to familiarize ourselves with the legal texts themselves, particularly their structure and characteristic properties. This enabled us to set objectives and tasks for subsequent research in the field of extracting and structuring knowledge from unstructured text.

We explored classical NLP techniques to assess their suitability for extracting structured ontological descriptions and definitions in the Czech language. This exploration involved evaluating various models and methods to handle the complexities of the Czech language's inflectional morphology. Based on the research of existing systems utilizing these methods and the active movement transitioning from classical methods to metalearning approaches using large language models, we decided to research the applicability of LLMs for our tasks. We tested and familiarized ourselves with both basic and advanced techniques, which we subsequently used to design our solution.

The experimental results indicate that for the specific requirements of our task, employing and adapting Large Language Model (LLM) methodologies yielded superior outcomes compared to traditional approaches. At the same time, we concluded that direct generation of source code is unsuitable due to the rich syntax of the formats. Furthermore, we tested an orchestration approach, which we subsequently applied in many of the designed OntoBuilder extraction components.

Based on the survey of state-of-the-art methods and techniques designed for natural language processing using large language models, and subsequent experiments with the Czech language, specifically Czech law, we proposed an algorithm or rather set of procedures, modifications of existing techniques, and developed an end-to-end collaborative system intended for domain experts for the extraction of terms, definitions, and subsequent ontology creation. A detailed description of the individual components for the subtasks of extraction, post-processing, and propagation in the form of classes and their descriptions into the ontology is above in Chapter 5.1 OntoBuilder.

Another major task of our work was to verify the functionality and performance of the proposed techniques. For this purpose, we created a web application designed for direct interaction with the user, a domain expert. It contains components for uploading and manipulating legislative documents. Additionally, it contains an interface for correcting and approving identified candidates for legislative terms, as well as for editing and correcting the definitions of approved terms. Lastly, it also includes supplementary components for the extraction of subclasses and properties. To enhance user experience, we integrated a visual representation of the created ontology in the form of a graph view and a preview of the source code. The entire process is parameterized, allowing the user to modify it via settings. The designed architecture significantly improved the quality of extraction by reusing examples from previously completed runs.

Through manual testing in the form of test scenarios provided to domain experts, we verified the quality of the proposed procedures and identified their shortcomings. The test results are presented in Chapter 6.2 Testing. Validation was conducted on multiple predefined scenarios and documents from the field of construction law and other contexts chosen by the testers themselves. During the development and our testing, we worked with parts of laws 183/2006 (Building Act), 128/2000 (Municipalities Act), 197/2004 (Fisheries Act), and 304/2013 (Public Registers of Legal and Natural Persons Act), which were at least partially processed within the Semantic Dictionary of Terms[22].

The very positive feedback from testers, whether regarding the quality of extraction, the simplicity of application usage, or the benefits of saving and accelerating their work, as well as pointing out shortcomings, motivated us to make improvements and conduct further research. Based on this, in Chapter 5.2 OntoAssistant, we proposed an additional alternative solution, which, unfortunately, has not yet been tested and is still in the experimental phase.

In future improvements, we aim to expand the expressivity and take advantage of using the OWL language. Specifically, we see potential in utilizing collaborative LLM reasoners for linking existing ontologies. Additionally, our goal is to integrate the OntoAssistant approach into the web application and proceed with the subsequent testing process.

Bibliography

[1] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623, 2021.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[4] Francisco Castillo. Embeddings: Meaning, examples and how to compute, feb 2023.

[5] Czech Telecommunication Office. Nařízení č. 10/2016. `https://ctu.gov.cz/sites/default/files/obsah/stranky/409167/soubory/narizenic.102016.pdf`, 2016.

[6] Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models secretly perform gradient descent as meta-optimizers. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4005–4019, 2023.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] Johannes Frey, Lars-Peter Meyer, Natanael Arndt, Felix Brei, and Kirill Bulert. Benchmarking the abilities of large language models for rdf knowledge graph creation and comprehension: How well do llms speak turtle?, 2023.

[9] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[10] Petr Křemen. Towards the web ontology language (owl), 2024.

[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[12] Feng Lin, Dong Jae Kim, Tse-Husn, and Chen. When llm-based code generation meets the software development process, 2024.

[13] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[14] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[15] Brandon McKinzie, Zhe Gan, Jean-Philippe Fauconnier, Sam Dodge, Bowen Zhang, Philipp Dufter, Dhruti Shah, Xianzhi Du, Futang Peng, Floris Weers, Anton Belyi, Haotian Zhang, Karanjeet Singh, Doug Kang, Ankur Jain, Hongyu Hè, Max Schwarzer, Tom Gunter, Xiang Kong, Aonan Zhang, Jianyu Wang, Chong Wang, Nan Du, Tao Lei, Sam Wiseman, Guoli Yin, Mark Lee, Zirui Wang, Ruoming Pang, Peter Grasch, Alexander Toshev, and Yinfei Yang. Mm1: Methods, analysis insights from multimodal llm pre-training, 2024.

[16] Microsoft. Taskweaver architecture, 2023.

[17] Microsoft. How to use plugins in semantic kernel, 2024.

[18] Microsoft. Overview of semantic kernel, 2024.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[21] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work?, 2022.

[22] Ministerstvo vnitra České republiky. Sémantický slovník pojmů. `https://opendata.gov.cz/dokumenty:sÃľmantickÃ¡-slovnÃŋk-pojmÅŕ:start`, 2020.

[23] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. Llm is like a box of chocolates: the non-determinism of chatgpt in code generation, 2023.

[24] Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. Language model tokenizers introduce unfairness between languages. *Advances in Neural Information Processing Systems*, 36, 2024.

[25] Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D Manning. Universal dependency parsing from scratch. *arXiv preprint arXiv:1901.10457*, 2019.

[26] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[27] Sumedh Rasal and E. J. Hauer. Navigating complexity: Orchestrated problem solving with multi-agent llms, 2024.

[28] Lama Saeeda, Martin Ledvinka, Miroslav Blaško, and Petr Křemen. Entity linking and lexico-semantic patterns for ontology learning. In *The Semantic Web: 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings 17*, pages 138–153. Springer, 2020.

[29] Aditi Singh, Abul Ehtesham, Saifuddin Mahmud, and Jong-Hoon Kim. Revolutionizing mental health care through langchain: A journey with a large language model. In *2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0073–0078. IEEE, 2024.

[30] Bea Stollnitz. Retrieval-augmented generation (rag). Bea Stollnitz's Blog, 2023. Available online: `https://bea.stollnitz.com/blog/rag/`.

[31] Alex Stolz, Bene Rodriguez-Castro, and Martin Hepp. Rdf translator: A restful multi-format data converter for the semantic web, 2013.

[32] Jana Straková, Milan Straka, and Jan Hajič. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[33] W3C. *RDF Vocabulary Description Language 1.0: RDF Schema.* World Wide Web Consortium, February 2004.

[34] W3C. OWL 2 Web Ontology Language Document Overview (Second Edition), 2014.

[35] W3C. RDF 1.1 Concepts and Abstract Syntax, 2014.

[36] Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, and Guoyin Wang. Gpt-ner: Named entity recognition via large language models, 2023.

[37] WisdomML. Nlp course in python: From beginner to expert in 2 months, April 2023.

[38] Fangzhou Wu, Xiaogeng Liu, and Chaowei Xiao. Deceptprompt: Exploiting llm-driven code generation via adversarial natural language instructions, 2023.

[39] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models, 2023.

[40] Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions, 2023.

[41] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Appendix A

A.1 Semantic Chunking Example

Original

ČÁST PRVNÍ

OBECNÁ ČÁST HLAVA I

PŘEDMĚT ÚPRAVY A JEJÍ ZÁKLADNÍ ZÁSADY Díl 1

Soukromé právo § 1

(1) Ustanovení právního řádu upravující vzájemná práva a povinnosti osob vytvářejí ve svém souhrnu soukromé právo. Uplatňování soukromého práva je nezávislé na uplatňování práva veřejného.

(2) Nezakazuje-li to zákon výslovně, mohou si osoby ujednat práva a povinnosti odchylně od zákona; zakázána jsou ujednání porušující dobré mravy, veřejný pořádek nebo právo týkající se postavení osob, včetně práva na ochranu osobnosti.

Extracted Chunk 1

ČÁST PRVNÍ OBECNÁ ČÁST HLAVA I PŘEDMĚT ÚPRAVY A JEJÍ ZÁKLADNÍ ZÁSADY Díl 1 Soukromé právo § 1 (1) Ustanovení právního řádu upravující vzájemná práva a povinnosti osob vytvářejí ve svém souhrnu soukromé právo. Uplatňování soukromého práva je nezávislé na uplatňování práva veřejného.

Extracted Chunk 2

ČÁST PRVNÍ OBECNÁ ČÁST HLAVA I PŘEDMĚT ÚPRAVY A JEJÍ ZÁKLADNÍ ZÁSADY Díl 1 Soukromé právo § 1 (2) Nezakazuje-li to zákon výslovně, mohou si osoby ujednat práva a povinnosti odchylně od zákona; zakázána jsou ujednání porušující dobré mravy, veřejný pořádek nebo právo týkající se postavení osob, včetně práva na ochranu osobnosti. This way if chunks 1 and 2 would be processed separately, the important information about chunk 2 belonging to ČÁST PRVNÍ etc is not lost.

A.2 Term Data Class

```
class Term(BaseModel):
    uri: HttpUrl
    label: Label
    definition: Optional[Definition] = None
    subTerms: Optional[List[SubTerm]] = None
    glossary: Optional[HttpUrl] = None
    vocabulary: Optional[HttpUrl] = None
    state: Optional[HttpUrl] = None
    types: Optional[List[str]] = None
    parentTerms: Optional[List[ParentTerm]] = None
    altLabels: Optional[List[Label]] = None
    hiddenLabels: Optional[List[Label]] = None
    description: Optional[Description] = None
    sources: Optional[List[str]] | Optional[str] = None
    notations: Optional[List[str]] = None
    examples: Optional[List[str]] = None
    exactMatchTerms: Optional[List[Dict[str, Any ]]] = None
    related: Optional[List[HttpUrl]] = None
    relatedMatch: Optional[List[HttpUrl]] = None
    properties: Optional[Dict] = None
```

This example shows the data structure from the TermIT system that is currently used to store all properties. For our work we only need a subset of the properties mentioned, but for the sake of linking and data exchange we try to use existing conventions.

A.3 Definition Candidate Examples

Input Chunk

ČÁST TŘETÍ - ÚZEMNÍ PLÁNOVÁNÍ
HLAVA III - NÁSTROJE ÚZEMNÍHO PLÁNOVÁNÍ
Díl 3 - Územně plánovací dokumentace
Oddíl 3
Územní plán
§43
(2) V územním plánu lze vymezit plochu nebo koridor, v němž je rozhodování o změnách v území podmíněno smlouvou s vlastníky pozemků a staveb, které budou dotčeny navrhovaným záměrem, jejímž obsahem musí být souhlas s tímto záměrem a souhlas s rozdělením nákladů a prospěchů spojených s jeho realizací (dále jen „dohoda o parcelaci“), zpracováním územní studie nebo vydáním regulačního plánu..."

Input Term

dohoda o parcelaci

Extracted Definition Candidate

Smlouva s vlastníky pozemků a staveb, které budou dotčeny navrhovaným záměrem, jejímž obsahem musí být souhlas s tímto záměrem a souhlas s rozdělením nákladů a prospěchů spojených s jeho realizací.

A.4 Example Output of Atomic Ontology Extraction

```
@prefix : <http://www.example.com/ontology/czech-legal#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:StavebníZáměr a owl:Class ;
    rdfs:label "stavební záměr"@cs ;
    rdfs:comment "Stavebním záměrem se rozumí podle okolností stavba,
        "změna dokončené stavby, terénní úprava, zařízení nebo údržba."@cs .

:Stavba a owl:Class ;
    rdfs:label "stavba"@cs ;
    rdfs:subClassOf :StavebníZáměr .

:ZměnaDokončenéStavby a owl:Class ;
    rdfs:label "změna dokončené stavby"@cs ;
    rdfs:subClassOf :StavebníZáměr .

:TerénníÚprava a owl:Class ;
    rdfs:label "terénní úprava"@cs ;
    rdfs:subClassOf :StavebníZáměr .

:Zařízení a owl:Class ;
    rdfs:label "zařízení"@cs ;
    rdfs:subClassOf :StavebníZáměr .

:Údržba a owl:Class ;
    rdfs:label "údržba"@cs ;
    rdfs:subClassOf :StavebníZáměr .
```