Utrecht University

Automatic software product features extraction from software vendor documents

Author:
Alexandros Constantinou 2126974

1st supervisor:
Dr. Slinger Jansen
2nd supervisor:
Dr. Fabiano Dalpiaz
External daily supervisor:
Dr. Siamak Farshidi

06-12-2024

Statement on the Use of Large Language Models (LLMs)

This document was created using standard academic tools. I hereby acknowledge the additional support from Large Language Models (LLMs), such as ChatGPT by OpenAI. The use of LLMs was limited to the following tasks:

- Structuring and organizing sections of the document.
- Generating templates for figures, tables, and technical layouts.
- Providing concise summaries of referenced studies and background materials.
- Generating code snippets for standard functions.

The use of LLMs was limited to these tasks and did not involve generating original research, interpreting results, or conducting analysis. All intellectual contributions, interpretations, and conclusions presented in this thesis are the author's own work.

Contents

1 Introduction

In the rapidly evolving field of software engineering, making well-informed architectural decisions is crucial for any software product to meet the system requirements. These requirements ensure the software fulfills its intended purpose and satisfies both user and stakeholder needs. Software production is a continuous decision-making process (Fitzgerald & Stol, 2014). Throughout the software development lifecycle, practitioners make critical decisions, such as selecting technology stacks. Decision support software has been developed across various fields, including software engineering (Rus et al., 2003) since the creation of software products, systems, and services often leads to intricate decision models and processes (Badampudi et al., 2018).

To facilitate better-informed decision-making in software development, extracting information on software features is pivotal in terms of choosing the right software components for a system. A feature refers to a trait or behavior of a software system that is noticeable to the end-user (Kang et al., 1990). Features play a crucial role in defining and conveying the similarities and distinctions of products among stakeholders. They also guide structure, reuse, and variation throughout the various stages of the software development lifecycle (Apel et al., 2013). However, manually extracting these features and qualities from vendor websites or review platforms is time-consuming and prone to errors. The reason behind this is the exponential growth of the software industry, leading to many software products and various vendors trying to differntiate themselves by introducing new features. This has resulted in a fiercely competitive and varied market, posing significant challenges for organizations in making strategic and well-informed software acquisition decisions.

Software products provided by vendors from various backgrounds are anticipated to have different strengths and weaknesses (Wei & Wang, 2004). Selecting a suitable vendor requires skilled personnel (Rani et al., 2022). This requires assessing each vendor's capabilities and aligning them with the software's architectural goals, which involves analyzing substantial qualitative data (Jadhav & Sonar, 2011).

A systematic framework is necessary to support decision-makers in analyzing and selecting software vendors (Rani et al., 2022). Choosing a software provider depends on specific selection criteria and the vendor's capabilities, with a strong emphasis on the quality characteristics of the software to be implemented (Chin & Fu, 2014). That is, apart from the specific behavior of a software product, the choice also involves evaluating its quality characteristics, such as maintainability, performance efficiency, compatibility, and security. Moreover, using quality models is a recognized method for assessing and managing software product quality (Miguel et al., 2014). Hence, it is logical also to gather information about the quality of each software product.

For information extraction from web sources, adapting to dynamic data is essential given the ever-changing nature of online content (Sarawagi, 2008). This is particularly true with software-specific information, as vendors update their services rapidly (Ma & Kauffman, 2014).

Even after collecting information on software features, integrating data from various heterogeneous sources presents a challenge. This involves identifying semantically related information that describes the same real-world concept in different ways (Bergamaschi et al., 2001). Several approaches have been explored to address this issue, such as Data Linking, which can be defined as a process that takes two sets of data as input and produces a collection of mappings between entities of the two sets as output (Ferrara et al., 2011).

This research aims to automate and combine the above-mentioned processes, developing a tool for software engineering, specifically in vendor selection, thereby aiding practitioners in their decision-making.

1.1 Problem statement

In software engineering, making well-informed decisions concerning software product selection is pivotal. Obtaining structured information about software features and qualities from readily available data sources, such as vendor documents or online reviews, can be helpful in terms of comparing different software, choosing technology stacks, and possibly informing feature-specific design choices. Moreover, it can simplify the distinction between the ever-increasing array of vendors with various functionalities and specific behaviors. In addition, the rate at which software features and capabilities are being updated or expanded introduces even more uncertainty if the information is manually extracted and the effort required is not cost-effective. One must have a very good understanding of the market to make safe comparisons between software products with similar characteristics. This requires an automatic way of extracting and updating information about these features for design or comparison purposes.

A self-updating knowledge graph that can automatically collect data from heterogeneous sources, standardize it for comparability, and properly extract and integrate the available information about different software could be very impactful. The limited currently available literature that tackles this challenge reveals the gaps in this area:

- **Inefficiencies in Manual Extraction**: Research by Fitzgerald & Stol (2014), Rus et al. (2003), and Apel et al. (2013) demonstrates that manually extracting software characteristics and qualities is a time-consuming process that is prone to mistakes, a situation worsened by the rapid growth of the software industry.
- **Lack of Comprehensive Tools**: Research by Bakar et al. (2015) and Y. Li et al. (2017) indicates a noticeable lack of support tools for automated feature extraction, which results in limited practical use.
- **Challenges in Data Integration**: Studies by Haris et al. (2020), Davril et al. (2013), and Bergamaschi et al. (2001) showcase the difficulties in semantically analyzing and integrating information that sometimes may describe the same concept differently across sources.
- **Need for Improved Automated Methods**: As suggested by Miguel et al. (2014) and Bing et al. (2016), there is a need for automated methods for feature extraction and quality assessment, particularly methods that capture the semantic relationships and the dynamic nature of software information.

This research aims to address these gaps by developing an artifact that combines NLP techniques for data integration, standardization, and semantic linking to create a self-updating knowledge graph about features of different software products. This tool will aid software practitioners in making more informed decisions during vendor selection.

1.2 Scope and Limitations

This research focuses on automating the extraction of information related to software features and qualities. Nevertheless, considering the scope and available resources, there are some constraints on the extent to which this may be accomplished. The accuracy of the retrieved information will depend on the quality and quantity of data obtained from online sources, considering data privacy. Moreover, using unsupervised techniques might introduce inaccuracies and difficulty in assessing the results.

1.3 Background

Software feature extraction has been studied from a variety of perspectives. These can provide insights to enhance our understanding of this field. This section examines relevant research, emphasizing its methods and relevance to our research. The mentioned studies have been selected from both manual search and the set of results from the initial search process described in section 2.2, in the scope of the chosen literature study approach. These selections aim to paint the picture in terms of the previous approaches to accomplish similar goals to automatically extracting information about software products.

Hybrid (involving both rule-based and machine learning) NLP techniques are widely used for feature extraction in Software Product Lines Engineering (SPLE), according to a systematic literature review (Bakar et al., 2015). This study showcased automated and semi-automatic feature clustering techniques from information retrieval and data mining. However, a notable deficiency in public support tools challenges the practical implementation of these approaches.

Another systematic literature review (Y. Li et al., 2017) focuses on extracting features from natural language documents. This work demonstrates how semantic analysis methods, including Latent Semantic Analysis and Semantic Role Labelling, can enhance the overall precision and completeness of the feature extraction procedure. Their examination of data sources shows that Software Requirements Specifications (SRS) are the most commonly used data source. Since product descriptions are readily available and include a wealth of information, they are also a widely used data source. However, as pointed out by the authors, full automation can be challenging and not practical. This is because there are situations where only manual communication of a particular degree of domain knowledge is possible.

The use of SRS documents is also supported by Haris et al. (2020). This study uses NLP approaches to automatically identify and extract required sentences from SRS documents using Sequence Part-of-Speech (POS) tagging patterns. Word dependency parsing rules are then used to extract features from these phrases. To compare the extracted sentences with the real requirement sentences, their method still depends on manual extraction.

In terms of using data from online community platforms, machine learning methods have been explored to identify software requirements from StackOverflow . According to a systematic review, topic modelling using Latent Dirichlet Allocation (LDA) is frequently used for this (Ahmad et al., 2020). The study highlights the difficulties in automatically extracting and classifying unstructured online text to distinguish between functional and non-functional requirements. There is also a discussion of common evaluation techniques, including recall and precision. Although machine learning has shown great potential, the research suggests issues that need to be worked out, like the requirement for common pre-labelled datasets. Moreover, the requirements were not further processed using sentiment analysis to indicate the degree to which they are satisfied by a particular software product, despite the fact that they are generally identified with some degree of success.

An alternate method is provided by Bakar et al. (2016), which concentrates on publicly accessible software descriptions such as product brochures and online software reviews. In this study, Feature Extraction for Reuse of Natural Language Requirements (FENL), a semi-automated approach, is presented. Using keyword occurrences from different noun-verb-adjective combinations, this approach involves finding phrases that can be software features. The method includes the manual creation of a feature model and is assessed using common metrics such as recall, accuracy, and F-Measure, demonstrating similar performance with related works.

Feature Models (FMs) are a popular formalism for representing and analysing the similarities and differences of software products, according to Davril et al. (2013). They use contrastive analysis to automatically extract domain-specific phrases from textual documents that can be found on marketing websites like SoftPedia[1] and online product repositories. The authors classify these terms as either common or variable features using a combination of linguistic features and machine learning classifiers, based on their distribution in various vendor brochures. Resolving the semantic links between the detected features is not mentioned, despite the fact that this research aims to automatically develop a feature model.

The framework created in (Bing et al., 2016) maps popular features to associated product attributes on the product description pages in addition to identifying these traits from customer reviews. A discriminative graphical model based on hidden Conditional Random Fields (CRFs) is used in the paper to overcome the lexical gap between customer reviews and product descriptions. The framework uses unsupervised methods and is intended to be domain-agnostic, which means it may be applied to a variety of different domains without the need for labelled training data. The available review content for each product and the challenge of integrating the information that has been retrieved are the limiting factors of this strategy.

A graphical model is also used by Wong et al. (2008), whose method tackles both extraction and normalization of product attributes from text fragments found in web pages. The Dirichlet mixture model used allows for an unlimited number of attributes to be discovered. The authors make use of both content and layout information about the web pages structure. This introduces some limitations due to the dependency on the layout format of web pages, which can vary significantly across various sites.

Finally, a more recent approach by Liu et al. (2018) proposes a hierarchical attention recurrent neural network (RNN) as an end-to-end model for information extraction from the web. The advantage of using an RNN is that context features are naturally incorporated. The authors suggest that this model can be trained with only a few labeled pages, which reduces the manual annotation effort required. This method encourages the effort of training a model without many training examples, however, the information extracted in this case is concerned only with a limited number of specific record attributes.

[1] https://www.softpedia.com/

As a result, the ability to generalize and adapt to varied datasets or domains could be limited, especially if the number of attributes to be extracted is not pre-determined.

1.4 Research Questions

In response to the constraints outlined in the problem statement, a main research question (MRQ), was formulated:

MRQ: *How can software practitioners be supported in their technology selection process with automatically updated knowledge regarding software products across different vendors?*

This research question has been split up into six sub-research questions (RQs) to guide the research:

- **RQ1**: *Which unsupervised machine learning methods have been explored for capturing and structuring knowledge about software products from vendors' online resources into a knowledge graph?*
- **RQ2**: *What are the key characteristics and capabilities of unsupervised machine learning methods used to extract knowledge from software vendors' resources?*
- **RQ3**: *How can the effectiveness and accuracy of unsupervised machine learning methods for extracting knowledge about software products be evaluated?*
- **RQ4**: *How can an automated system be designed to extract and structure knowledge about software products from different vendors into a unified knowledge graph?*
- **RQ5**: *How can the performance and quality of a system for automatically generating a knowledge graph from software vendors' data be evaluated?*

Research Questions 1 to 3 are answered using a systematic literature review (SLR) and are related to the state-of-the-art information extraction methods described in the literature. This process involves extracting and analyzing data to generate insights into existing models, their attributes, and the methods used to evaluate them. Based on the findings from the literature review, Design Science methodology will be employed to develop and evaluate an automated software feature extraction system, providing answers to Research Questions 4 and 5. Additionally, experiments will be conducted to validate the methods used.

1.5 Baseline Pipeline

The ultimate objective of this research, which encompasses the development of an automated software feature extraction system, is visually represented in a baseline pipeline as seen in Figure 1.1. This pipeline was based on the approach proposed by (Farshidi & Zhao, 2022) and the current design incorporates two parallel pipelines to address different aspects of software feature extraction. The ***Software Features Extraction Pipeline*** focuses on extracting boolean feature information from vendor websites. The ***Contextual Data Extraction Pipeline*** aims to gather contextual data from external software engineering resources. The output from both pipelines is then integrated to form a comprehensive Knowledge Graph, providing a more holistic view of software features and qualities.

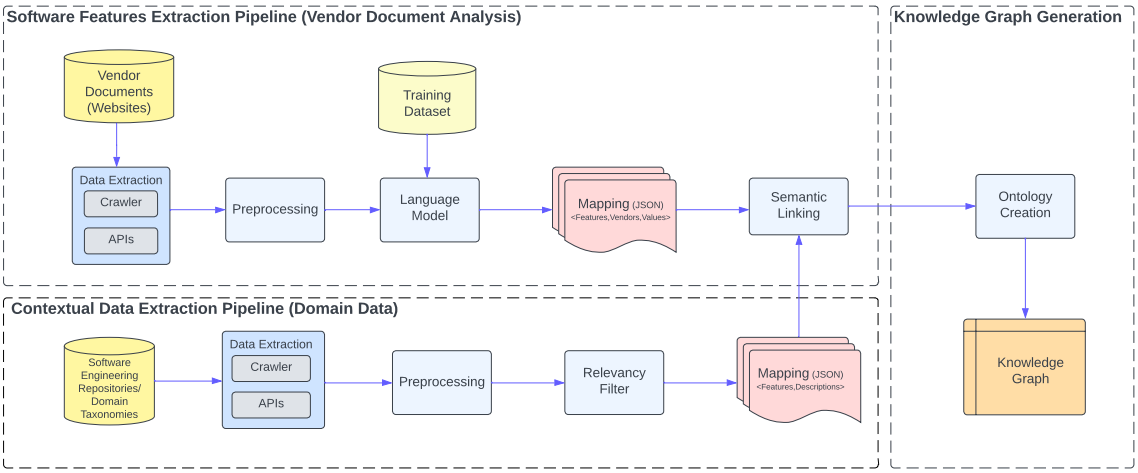

Figure 1.1: Baseline Pipeline of the Software Features Extraction Process

Vendor Documents is the primary data source for the Software Features Extraction Pipeline. It consists of various vendor websites that provide detailed information about software features, capabilities, and specifications. The content from these websites is scraped or accessed via APIs to serve as raw data for feature extraction.

Software Engineering Repositories/ Domain Taxonomies serve as the data sources for the second pipeline. It focuses on gathering contextual data from online software engineering resources. From such resources, more information can be extracted regarding software features which might not be directly accessible from vendors websites. This can aid the derivation of relationships between the identified features and consequently the creation of a domain ontology.

Data Extraction is part of both pipelines and entails the use of web crawlers or application programming interfaces (APIs) to systematically collect textual data, which is then stored in a structured format to facilitate its analysis.

Preprocessing is essential in both pipelines for cleaning and structuring the raw data collected. It focuses on preparing the textual data, removing redundant content, and converting it into a suitable format for the specific learning approaches used in the respective pipelines. The goal is to create a uniform dataset that can be effectively analyzed in the following stages.

The ***Training Dataset*** which is yet to be determined, serves as a resource for training and testing the feature extraction model. The dataset is expected to include a categorized list of software technologies. The listed features can also act as seed words for unsupervised learning methods like topic modeling, guiding the extraction process and improving the quality of the newly identified features.

The ***Language Model*** component, often leveraging advanced architectures like BERT (Devlin et al., 2019), serves as the core engine for feature extraction in the pipeline. It is trained to identify and extract relevant software features from vendor websites. The model understands the contextual relevance of words and phrases, enhancing its performance in various NLP tasks. This component is crucial for both semi-supervised and unsupervised learning approaches, enabling the system to generalize well to new, unseen data.

Mapping generates individual JSON files for each software vendor. These JSON instances serve as an intermediate representation of the extracted data and list all the software vendors against a unified set

of features, along with the boolean values indicating which features are offered by each vendor. This serves as the preamble knowledge based on which the knowledge graph will be constructed.
The ***Relevancy Filter*** is the component responsible to process the collected information and filter out noise, to ensure that the data used in subsequent stages of the pipeline is relevant.
The ***Mapping*** in this pipeline is responsible for organizing the feature information into a structured JSON format, where each feature will be listed along with descriptive information to provide a better understanding of what each feature represents.
The ***Semantic Linking*** component is responsible for identifying and resolving features that are semantically equivalent but may be expressed differently across various software-specific sources. This is crucial for consistently aggregating data, ensuring that the same features are not treated as different due to variations in terminology or phrasing.
Ontology Creation processes the outputs from both pipelines to create the domain ontology, based on which the ***Knowledge Graph*** will be structured. The resultant knowledge graph stores all the collected information, creating a comprehensive repository that intuitively displays the features offered by various software vendors.

2 Research Approach

This section outlines the diverse research methodologies employed in this study, which aims to automate the extraction of software features from vendor websites and review platforms. The research leverages a multi-faceted approach that includes a comprehensive literature review, design science research (DSR) for artifact creation, and data analysis.

2.1 Research Methods

The research questions outlined in Section 1.4 are tackled through a multi-faceted research strategy. This blend of methodologies enriches our understanding of the issues at hand, thereby strengthening the credibility and robustness of the findings. Our research leverages three primary methodologies: Literature Study, the application of Design Science principles, and empirical Experiments. Figure 2.1 offers a detailed breakdown of how each methodology contributes to answering our Main Research Question (MRQ) and Research Questions 1 through 5. A marked "X" at the cross-section of a methodology and a research question signifies that the methodology is employed to address that specific question.

Research Questions		Research methods		
		Literature study	Design Science	Experiments
MRQ	How can software practitioners be supported in their technology selection process with automatically updated knowledge regarding software products across different vendors?	X	X	X
RQ1	Which unsupervised machine learning methods have been explored for capturing and structuring knowledge about software products from vendors' online resources into a knowledge graph?	X	X	
RQ2	What are the key characteristics and capabilities of unsupervised machine learning methods used to extract knowledge from software vendors' resources?	X	X	
RQ3	How can the effectiveness and accuracy of unsupervised machine learning methods for extracting knowledge about software products be evaluated?	X	X	
RQ4	How can an automated system be designed to extract and structure knowledge about software products from different vendors into a unified knowledge graph?		X	X
RQ5	How can the performance and quality of a system for automatically generating a knowledge graph from software vendors' data be evaluated?		X	X

Figure 2.1: Research methods used to answer research questions

2.2 Literature study

The literature review serves as a foundational element of this research, focusing on the existing body of work related to software feature extraction, unsupervised NLP techniques, and decision support in software engineering. The aim is to identify gaps and opportunities in the current research landscape, thereby shaping the primary research questions of this study. Criteria for the inclusion and exclusion of academic papers have been established, focusing on their relevance to information extraction, knowledge graph construction, recommendation systems, and other related areas. In order to conduct the literature study, the protocol presented in (Farshidi et al., 2024) was followed.

The cornerstone of this research is the Systematic Literature Review (SLR), which serves to rigorously investigate and synthesize existing studies in the realm of automatic software feature extraction and unsupervised NLP techniques. The SLR is guided by a well-defined review protocol proposed by (Kitchenham, 2004), which outlines the methodological approach to be followed and consists of eleven phases: Problem formulation, research questions, review protocol (search strategy), search process, searching, screening, inclusion/exclusion criteria, quality assessment, data extraction, data analysis and synthesis, and reporting.

The **problem formulation** and **research questions** guiding this study are essential for a successful SLR, and they were outlined in the first chapter to establish the scope and context of the research. Each

of the subsequent stages was meticulously documented in separate spreadsheets to keep track of the progress.

The SLR methodology for this study is outlined in the **review protocol**, which follows a five-step approach based on the framework suggested by (Kitchenham, 2004). The initial step involves the search and screening process; the second step sets and applies criteria for inclusion and exclusion; the third step assesses the quality of the chosen papers; the fourth step gathers data from these papers; and the fifth step involves the analysis and synthesis of the collected data. The results of the outlined methodology can also be viewed in a spreadsheet, made available online[2].

The initial **search process** stage involved a manual search for relevant academic papers in the chosen field, guided by an initial hypothesis and understanding of the subject matter. An exploratory search was conducted in the Scopus database[3], based on initial hypotheses. These hypotheses were centered around key themes such as "software feature extraction", "unsupervised information extraction" and "decision support in software engineering". The initial set of assembled papers served as the basis for deriving a search term for a more extensive, automated **searching** process. Armed with this search term, the study then expanded its focus to four major digital libraries: IEEEXplore [4], Springer[5], ACM DL[6], and ScienceDirect[7]. All the gathered papers were then exported into a CSV file and information on each is thoroughly documented in a spreadsheet, capturing the following details: Title, Authors, Abstract, Keywords, Venue, Venue Ranking or Quality as documented in Scimago Journal and Country Rank (SJR)[8] for academic journals and in the Computing Research & Education (CORE)[9] Conference Portal for conference papers, Citations count, Year of publication. During the **screening** phase, a preliminary assessment of the gathered papers was conducted to determine their relevance to this research. This assessment involved a quick examination of the papers' abstracts, keywords, and any other pertinent details. Hence, the Relevance was also recorded using ordinal values (None, Low, Medium, High) to append to the aforementioned details.

Subsequent to this initial screening, **inclusion and exclusion** criteria were employed to separate relevant papers from those that were not pertinent. Criteria for consideration included the paper's language (English), availability, relevance score, year of publication, citation count, and the prestige of the conference or journal where it was published. After considering these factors, each paper was assigned a score and a predefined score threshold was then applied to finalize the selection of papers for inclusion in the SLR.

Following the inclusion/exclusion process, a **quality assessment** of each chosen paper was carried out. This assessment involved extracting relevant information from the papers, including details about the research methodology (whether it constituted a literature study or an experimental study, the use of qualitative or quantitative methods etc.), data collection approach, and the authors' chosen evaluation method. Moreover, it involved inspecting whether the paper presented a clear problem statement, research questions, research challenges, findings, and real-world use cases. This additional set of standards was established to evaluate each paper's quality further and determine their incorporation in the study.

Papers that met these quality standards were then subject to **data extraction**, where key information like utilized features, models, and evaluation methods were gathered. This data served as the basis for understanding current best practices in the field.

The final stage involved **data analysis and synthesis**, aimed at understanding the collected information, filtering out less popular models, and categorizing similar features under unified terms. The culmination of this SLR is a comprehensive report detailing the findings, which is elaborated upon in the third chapter of this study.

[2] https://zenodo.org/records/13899562
[3] https://www.scopus.com/home.uri
[4] https://ieeexplore.ieee.org/Xplore/home.jsp
[5] https://link.springer.com/
[6] https://dl.acm.org/
[7] https://www.sciencedirect.com/
[8] https://www.scimagojr.com/
[9] http://portal.core.edu.au/conf-ranks/

2.3 Design science

This research utilized a Design Science methodology, as outlined by (Hevner, 2002). Tailored to facilitate inquiry in the Information Systems domain, the primary aim of Design Science research lies in the development of innovative artifacts and the elucidation of methodologies for their creation, thereby enhancing the existing environment. Within the scope of Information Systems, such artifacts may include constructs, models, methods, implementation, and algorithms. Three fundamental cycles constitute the framework of Design Science research. The initial cycle referred to as the *Relevance Cycle*, focuses on the applicability of the research within its designated context. This cycle delineates both the research prerequisites and the evaluation metrics, provoking essential queries regarding the artifact's contribution to environmental betterment and the quantification of such improvements.

Subsequently, the *Rigor Cycle* posits that a substantial knowledge base, comprising scientific theories and engineering methods, underpins Design Science research. This foundational layer facilitates rigorous exploration within the research domain. The knowledge base should incorporate state-of-the-art expertise, prior experiences in the research field, and pre-existing artifacts and processes within the application domain.

The final cycle, known as the *Design Cycle*, serves as the pivotal mechanism that governs the entire process. Activities within this cycle center on the artifact's creation and evaluation, with immediate feedback guiding subsequent design refinements. Although this phase represents the research's focal point, a reasonable equilibrium between artifact development and its assessment is imperative.

Integration of these three cycles results in the creation of an artifact that maintains a robust standing within the research area and endures scrutiny across both theoretical and practical dimensions of the research domain. Figure 2.2 delineates the Design Science model, encapsulating the aforementioned cycles.

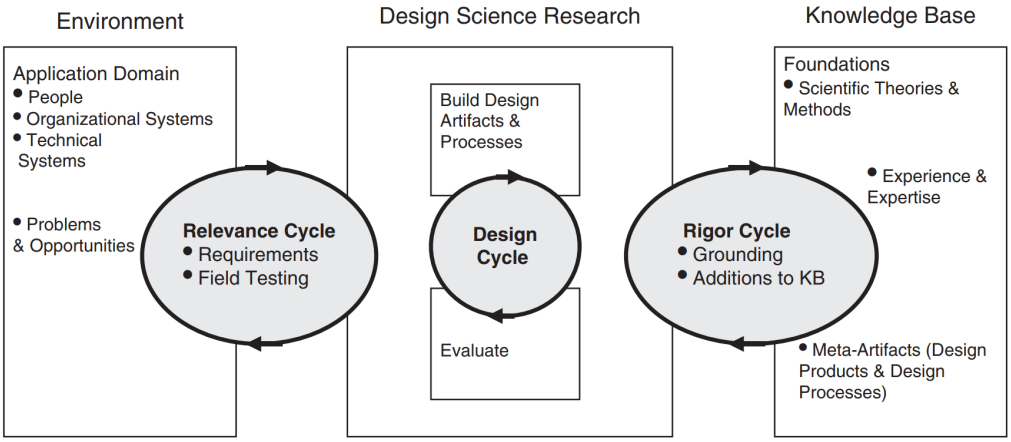

Figure 2.2: Designed Science Model based on (Hevner, 2002)

3 Systematic Literature Review

This chapter presents the findings of the systematic literature review (SLR) outlined in Chapter 2. The SLR aimed to acquire an in-depth understanding of automatic software feature extraction, focusing on techniques and models employed. The review also sought to address specific research questions (RQ1-3) pertaining to using unsupervised NLP techniques for extracting software features from vendor websites, documentation, and review platforms. This includes the machine learning models, techniques, and algorithms used in existing literature, as well as the sub-tasks and methods they employ, such as text processing methods, language modeling, and other aspects of information retrieval. The collected data serves as a foundation for designing and implementing the feature-extracting artifact proposed in this research.
The SLR was conducted following the guidelines proposed by (Kitchenham, 2004). Additionally, the systematic literature review performed by (Farshidi et al., 2020) served as a valuable reference for the methodology and scope of this SLR. The subsequent sub-chapters will delve into the process, rationale, and key findings of the SLR. Each stage of the conducted literature study was documented in a spreadsheet which is made publicly available (Constantinou, 2024).

3.1 Data Sources and Search Strategy

Chapter 2 outlined the methodology for the literature search, which was executed in two distinct phases: initial hypothesis and automatic search. The initial hypothesis helped assemble an initial collection of academic papers, the keywords of which were then synthesized into a search term for automated data collection. Further details on this search mechanism will be discussed in the forthcoming section.
For this literature review, the main digital libraries used were the ACM Digital Library, Springer Publishing, IEEE Xplore Digital Library, and ScienceDirect. These platforms were selected for their credibility and the high academic standing of the papers they host, especially in the domain of interest.

3.2 Search Process

Figure 3.1 illustrates the sequence of actions taken and the cumulative count of papers obtained during the entire literature search process. This includes both the initial hypothesis and the automatic search stages. In the initial hypothesis stage, a set of papers was gathered based on existing domain-specific knowledge and theoretical foundations. To align with the research questions, initial search queries were formulated and executed in Scopus to retrieve pertinent academic papers, such as:

- “unsupervised feature extraction”
- “software documentation” AND (“text mining” OR “information extraction”)
- “websites” AND (“text mining” OR “information extraction”)
- “software engineering” AND (“decision-making” OR “technology selection”)
- “feature extraction” AND “language models”
- “information retrieval" AND “textual data”

The details of the 73 extracted papers, i.e., the Title, Authors, Abstract, Keywords, Venue, and Venue Ranking, were documented, as detailed in Chapter 2. The keywords of these papers were then leveraged to create the search term to be used in the automatic search. The utilized search term is presented below:
(“software feature” OR “product feature” OR “requirements” OR “software selection” OR “knowledge”) AND (“textual” OR “web” OR “text mining”) AND (“unsupervised”) AND (“extraction” OR “retrieval”)

The search term was designed to capture current methodologies and key terms in automatic software feature extraction, thereby streamlining the acquisition of papers from the previously mentioned digital libraries. The outcomes of this automatic search were saved in either CSV or Bibtex formats, offering a streamlined approach for organizing and storing these documents. Subsequently, duplicate, incomplete, or corrupted entries were eliminated. Notably, the focus was mainly on papers published after 2016, although some older yet impactful papers were also included at different stages. The automatic search phase yielded 2131 papers, resulting in a comprehensive collection of 2204 papers from manual and automated search methods.

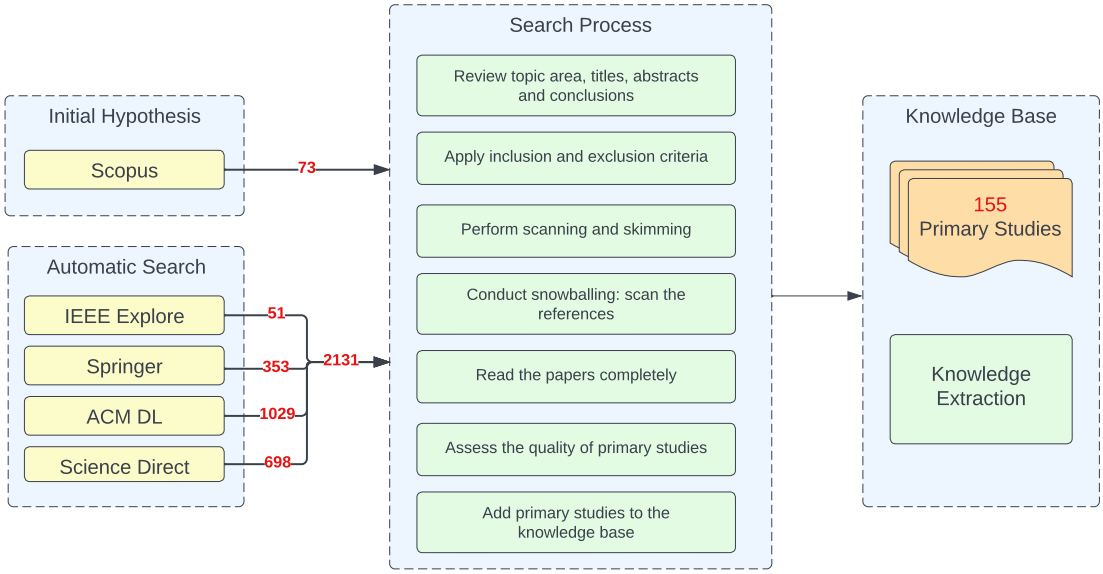

Figure 3.1: SLR Search Process based on (Farshidi et al., 2020)

Afterwards, a multi-step approach was employed to ensure comprehensive and quality data collection. Initially, the topic area of the collected papers was reviewed, followed by a thorough examination of titles, abstracts, and conclusions of potential papers. Inclusion and exclusion criteria were then applied to filter the papers further. Scanning and skimming techniques were used for a quick assessment of the content. The snowballing method was also implemented, wherein the references of selected papers were scanned for additional relevant studies. Subsequently, the remaining papers were read to evaluate their quality and relevance to the research questions. High-quality primary studies were then added to the knowledge base for further analysis. The final count of papers subject to analysis and data extraction was 155.

3.3 Inclusion and Exclusion Criteria

After completing both the manual and automated search phases, an assessment stage was initiated to scrutinize the compiled literature, including papers obtained through the snowballing technique. In this stage, the abstracts and keywords of each paper were meticulously examined to determine their relevance to the study on automatic software feature extraction. Papers were then ranked on an ordinal scale with four categories: None, Low, Medium, and High. A “None” ranking indicated that the paper had no relevance to the study, essentially deeming it out of the review process, while a “High” ranking suggested significant relevance.
Due to the emerging nature of the research field, fewer papers are highly relevant, making relevance a more heavily weighted factor in the selection process. Citation count and venue ranking bore equal weights, while the year of publication was used to prioritize more recent methods, which are more likely to represent the current state of the art.
Assessing these factors for each study indicated their potential for inclusion in the next phase of literature selection. Nevertheless, a definitive decision could not be made solely based on this score, as the initial screening lacked sufficient information to determine relevance conclusively. It is essential also to note that more recent studies may not have a high number of citations at the time of collection of papers, but this is not always indicative of their practical value.

3.4 Quality Assessment

Following the inclusion and exclusion criteria implementation, a further quality assessment of the selected papers was conducted. This assessment acted as a more in-depth extension of the initial criteria, offering a thorough evaluation of each paper in the context of automatic software feature extraction. Each

paper underwent a structured review and annotation process, wherein quality assessment questions were answered with binary outcomes: "Yes" or "No". For example, if a paper had a clear problem statement, the corresponding field in the evaluation data sheet was marked as "Yes" for that particular paper. Apart from the problem statement, the evaluation criteria also included: Inclusion of research questions, clear research challenges, clear statement of findings and finally, whether the results of the study had real-world use cases or not.
As previously noted, a total of 155 studies were ultimately chosen for data extraction. Before exploring the data extracted from these studies, it's essential to consider the academic venues where they were published. Figure 3.2 visually represents the distribution of these selected papers across various scholarly journals. This visualization not only underscores the range of sources but also adds a layer of credibility to the thoroughness of the literature review. It offers a quick overview of the academic settings that contribute to the foundation of this research. Figure 3.3 indicates the count of journal papers and conference papers and the distribution of scores assigned to them by the online platforms mentioned in Chapter 2. These figures underscore the range of sources but also add a layer of credibility to the thoroughness of the literature review.

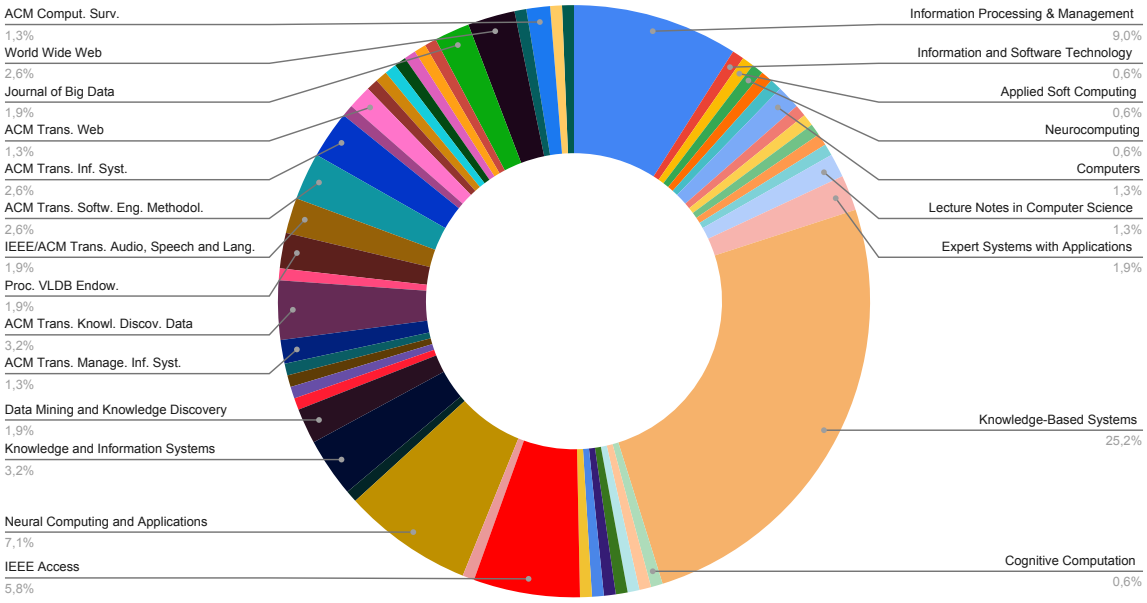

Figure 3.2: Distribution of Publication Journals

Journal Papers		Conference Papers	
SJR Rank	Count	CORE Rank	Count
Q1	129	A*	2
Q2	18	A*	3
		B	1
		N/A	2
	147		8
			Total = 155

Figure 3.3: Overview of the rankings of the selected studies

Apart from the academic venues in which they are published, the temporal distribution of the selected studies also holds importance. The primary emphasis of this literature review has been directed towards research articles published after 2016. This deliberate selection is based on the rationale that such papers are more inclined to represent the current state of the art. This is reflected in Figure 3.4, which shows the year-wise distribution of the selected papers. The data reveals that the predominant articles were published in 2020 and 2021, amounting to 35 and 29 papers, respectively. In both 2022 and 2023, there is a notable contribution of 22 papers. Although the inclusion criteria assigned a lower score to older

publications, several influential works dating back to 2009 have been included due to their enduring significance in the area. This chart offers a detailed overview of the period covered by the literature that serves as the foundation for this research.

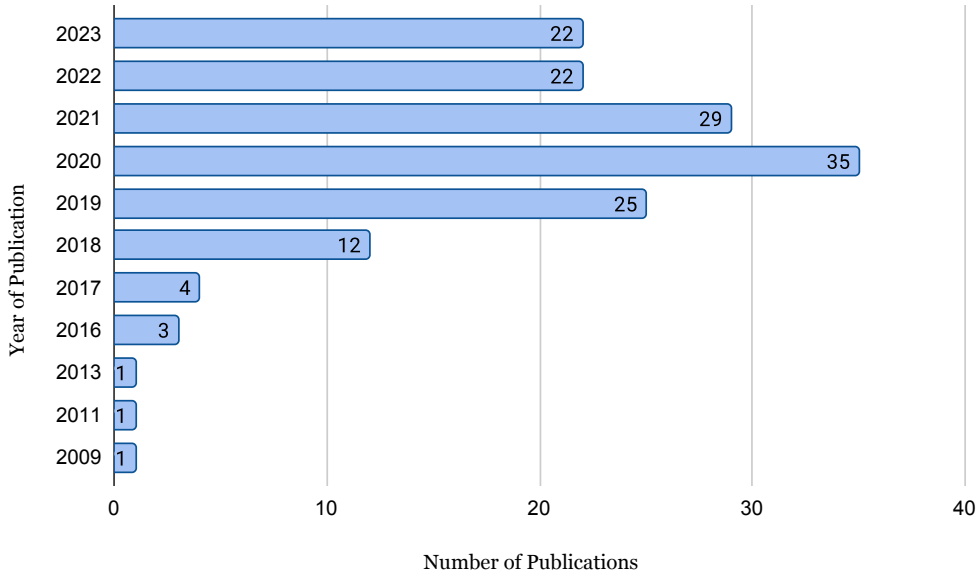

Figure 3.4: Yearly distribution of the selected publications

3.5 Data Extraction

After applying the inclusion and exclusion criteria and assessing the quality of the articles, a final selection of papers suitable for data extraction was made. This marked the beginning of the data extraction phase. The primary objective was to methodically gather and categorise relevant information from the chosen articles to assist a thorough comprehension of automated software feature extraction.

3.5.1 Models, Techniques & Algorithms

The relevant data for this systematic literature review (SLR) encompassed the machine learning models, methods, algorithms, and assessment approaches utilized. Distinctive attributes and qualities of diverse implementations in different areas and applications were also documented. These characteristics define other aspects of each implementation, covering essential components such as data representation and pre-processing and more complex and specialised properties. These encompass various educational approaches and elements such as sentiment lexicons, complex analytical tasks, model-specific setups, and other specialised methods. This material provides insights into the layered complexity of machine learning implementations, spanning from fundamental components to advanced features. It offers a structured approach to comparing the diverse range of techniques and methods used in the literature.

3.5.2 Tools, Libraries & Resources

The various tools, libraries, and resources identified in the academic papers span multiple NLP and machine-learning areas. These resources can be helpful in different aspects of the design. For instance, some tools specialized in text processing and natural language understanding can aid the extraction of pertinent information from the raw textual data, particularly identifying segments that contain information related to software features.

3.5.3 Datasets

Numerous studies in the literature have utilized existing datasets for their research. The extraction of these datasets could offer insights for developing the feature extraction artifact, specifically in terms of the features and subtasks they encompass. While most identified datasets are not explicitly tailored

for unsupervised software feature extraction, they could inform different aspects of the artifact's design, such as feature engineering, data pre-processing, and evaluation metrics. Moreover, these datasets cover a multitude of NLP tasks, such as sentiment analysis and text classification, which could be incorporated into the pipeline introduced in Chapter 2 to improve its robustness and versatility.

3.5.4 Research Annotations

In addition, administrative details were also collected, including the name provided to each approach by the authors (if any), the type of research, the type of learning implemented (supervised, unsupervised, etc), the domain category (such as text mining, classification), the specific application (such as sentiment analysis, keyphrase extraction etc), the data collection method (manual or automatic), and any available GitHub URLs or other platform URLs related to the approach.

3.6 Data Analysis and Synthesis

This section involves a comprehensive examination of the knowledge acquired from the SLR. The goal serves a dual function: to guide the design decisions about the forthcoming feature extraction artifact and to address the initial research questions. An exhaustive component analysis is conducted, examining the occurrences of models and investigating temporal patterns in their utilization. In addition, this study examines the relevant application areas and implementation characteristics that are important for the artifact's design. This comprehensive study is the foundation for comprehending the range of machine learning models, techniques, and applications that might be included in our feature extraction system. By linking these insights with potential evaluation methods, a holistic view is provided to guide the creation and evaluation stages of the artifact.

3.6.1 Component Analysis

The component analysis is a crucial result of the SLR and establishes the foundation for the engineering and design stages of the artifact. The requirements indicated in the conceptual pipeline will influence the design of the feature extraction system. This research seeks to uncover opportunities for significant contributions by analysing the methodologies utilised in state-of-the-art models and methods. The study yields a strong quantitative knowledge foundation that addresses the primary research questions.
Initially, the component analysis started with pre-processing data to exclude models with a lower frequency of occurrence in the literature. By explicitly considering models that appear four or more times, the study was refined to include 51 models with a significant presence in academic research. This guarantees that the models chosen for further analysis are extensively researched and likely to be dependable and efficient for various tasks. Table 3.5 provides a concise overview of this component analysis, emphasising the frequency of different models and possible combinations of methodologies documented in the current body of research.
Before discussing the clustering of models, it is sound to mention a few notable co-occurrences. BERT and BiLSTM appear together 17 times, as do GloVe and BiLSTM. The significant co-occurrence counts suggest a synergistic association between these models, commonly used with intricate tasks in natural language processing. Additional notable pairs include Word2Vec and Cosine Similarity, which have 11 co-occurrences, and CRF and BiLSTM, which have 10. These common pairings serve as a foundation for future clustering studies to provide a more comprehensive knowledge of how models might be efficiently matched or grouped for different tasks. Below, we will explore several noteworthy pairings of co-occurrences.
(BERT, BiLSTM): The frequent co-occurrence of BERT (Devlin et al., 2019) and BiLSTM (Huang et al., 2015) suggests a combination of pre-trained contextual embeddings (BERT) with sequential modeling (BiLSTM). This pairing is particularly effective in tasks requiring contextual understanding of language and sequence processing.
(GloVe, BiLSTM): The combination of GloVe (Pennington et al., 2014) word embeddings and BiLSTM recurrent networks is often employed in tasks involving natural language understanding and generation. This pair is effective for sentiment analysis (Pimpalkar & Raj R, 2022).
(Cross Entropy Loss, BiLSTM): Cross Entropy Loss (Mao et al., 2023) is a common loss function used with BiLSTM and other Neural Network models in various classification and sequence-to-sequence tasks, including machine translation (H. Li & Lu, 2021).
(Word2Vec, Cosine Similarity): Word2Vec (Mikolov, Chen, et al., 2013a) embeddings paired with Cosine Similarity are widely used for measuring semantic similarity between words or documents. This

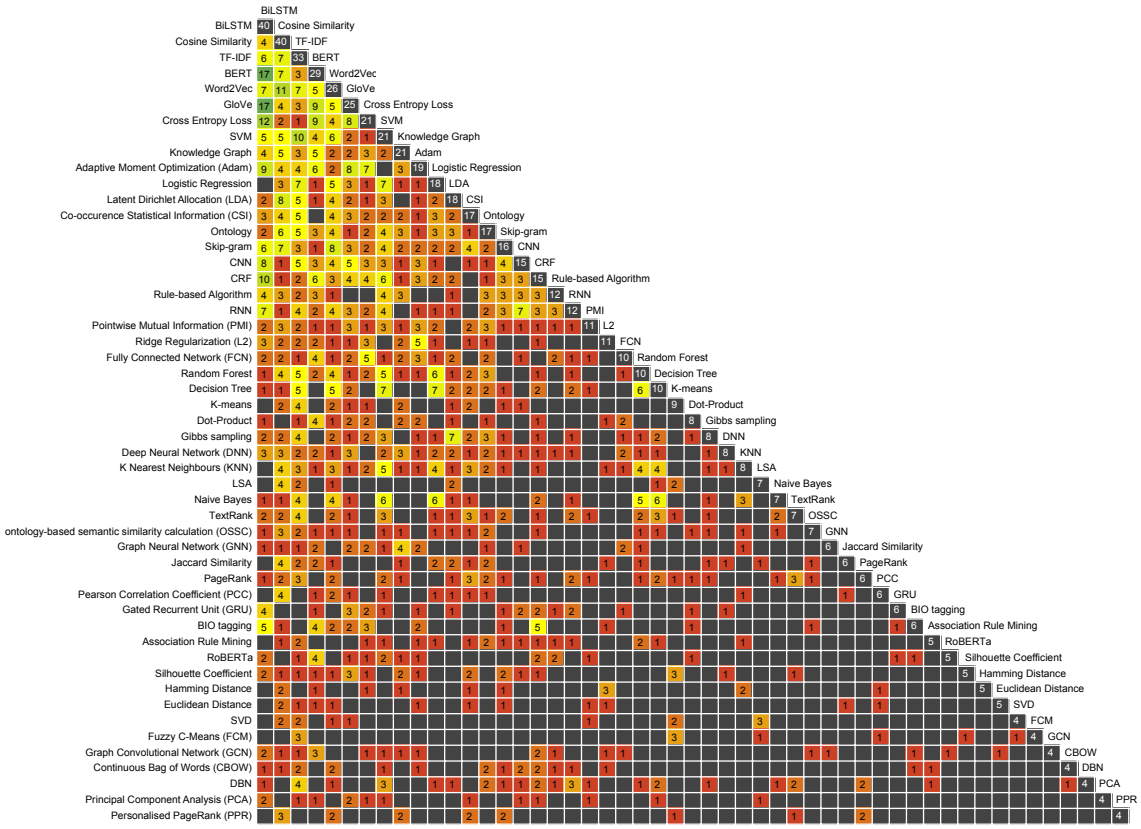

Figure 3.5: Model combinations occurrences found in literature

combination is foundational in information retrieval, recommendation systems, and document clustering, which may also be applied in the knowledge graph generation.

(CRF, BiLSTM): Conditional Random Fields (CRF) (Sutton & McCallum, 2010) and BiLSTM often co-occur in sequence labeling tasks (Huang et al., 2015). Such tasks include named entity recognition (NER) and part-of-speech tagging (POS), which are also often tackled in information extraction pipelines. K-Means clustering was employed to uncover patterns within these 51 models based on their co-occurrence in the literature. The method is particularly useful for several reasons. Firstly, it highlights commonly used models, offering insights into potential model compatibility for different functionalities in the feature extraction system. Secondly, it facilitates an understanding of the diversity of models available for different tasks, thereby enriching the design possibilities for the artifact. The elbow method was used to determine the optimal number of clusters, which is crucial for the interpretability and utility of the clustering results. This method revealed that 4 clusters provide a balanced trade-off between explanatory power and model complexity, however the first cluster was significantly larger than the others and was therefore subdivided into smaller, more focused clusters. Figure 3.6 is a culmination of the previous steps designed to uncover relationships between different machine learning models frequently employed together. A provisional interpretation of the thematic focus of each cluster is also shown based on the models it contains.

3.6.2 Temporal Trends

The subsequent research expands the investigation to temporal trends, examining the yearly distribution of these models from 2016 to 2023, as shown in Figure 3.7. This data provides insight into both the persistent significance of specific models and the changing patterns in their use over time. It is worth mentioning that models like BiLSTM, Cosine Similarity, and TF-IDF (Das et al., 2023) are regularly favored options, with BiLSTM appearing 39 times, Cosine Similarity showing 33 times, and TF-IDF appearing 31 times throughout the observed years. Conversely, models like Skip-gram (Mikolov, Sutskever, et al., 2013), Decision Tree (Rokach & Maimon, 2005), Gate Recurrent Unit (GRU) (**?**), and Hamming Distance (Bookstein et al., 2002) have declined usage, possibly reflecting shifts in technology or research

Cluster	Models	Interpretations
0	CRF, Dot-Product, GRU, BIO tagging, RoBERTa, GCN, CBOW	Focused on sequence processing and tagging techniques.
1	Cross Entropy Loss, Knowledge Graph, Adam, Ontology, Skip-gram, CNN, Rule-based Algorithm, PMI, L2, FCN, Gibbs sampling, DNN, GNN, Association Rule Mining	Geared towards optimization and semantic understanding through graph-based models.
2	LDA, K-means, LSA, Jaccard Similarity, PCC, Silhouette Coefficient, Hamming Distance, Euclidean Distance, SVD, FCM, PCA	Commonly used for topic modeling and clustering methods.
3	CSI, RNN, TextRank, OSSC, PageRank, DBN, PPR	Primarily involves ranking and information retrieval algorithms.
4	BiLSTM, BERT, GloVe	Focused on deep learning models for natural language understanding.
5	Cosine Similarity, Word2Vec	Primarily involves similarity and distance measure techniques.
6	TF-IDF, SVM, Logistic Regression, Random Forest, Decision Tree, KNN, Naive Bayes	Geared towards text classification and feature extraction.

Figure 3.6: Results of K-Means Clustering on the models

focus. These temporal insights offer additional contextual depth, assisting in comprehending model suitability and research trends.

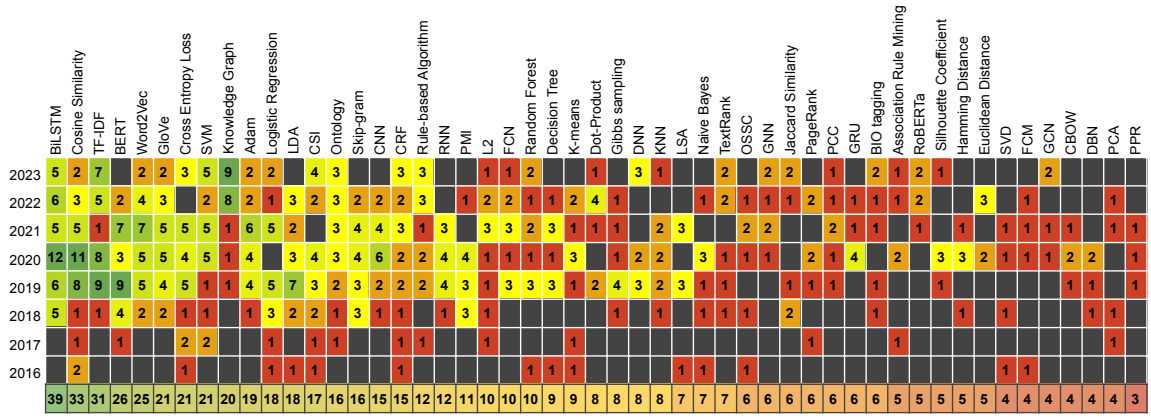

Figure 3.7: The trend of models mentioned in the literature over publication years, highlighting the popularity and emergence of various models

3.7 Model Categories

This section examines the often-mentioned types of machine learning models in the literature. Figure 3.8 categorizes these models into specific kinds, architectures, and architecture types. This categorization enables a deeper comprehension of the fundamental techniques, features, and computational structures that are often used in the field. These insights provide a detailed plan for making well-informed decisions on the design and implementation of the feature extraction artifact.

By thoroughly examining prominent model categories, architectures, and types of architecture, several significant observations arise. Models that quantify similarity or involve statistical reasoning are often used, indicating their potential usefulness in comparing and rating software characteristics. Optimization will likely be necessary in the artifact's design, either for selecting features or tweaking parameters. Language models have the potential to be highly beneficial in comprehending textual data, as the data also indicates the presence of a scoring mechanism. Chosen architectures are predominantly neural network-based solutions, highlighting the advantage of employing deep learning techniques for complex data relationships (Schmidhuber, 2015). Embedding models are crucial for transforming textual input into a format that is compatible with machine learning models. Graph-based techniques in architecture emphasise the significance of capturing interconnections between features, which may originate from phrase patterns and references to these features. The design of the feature extraction artifact matches state-of-the-art practices well by utilising these widely used methodologies and structures.

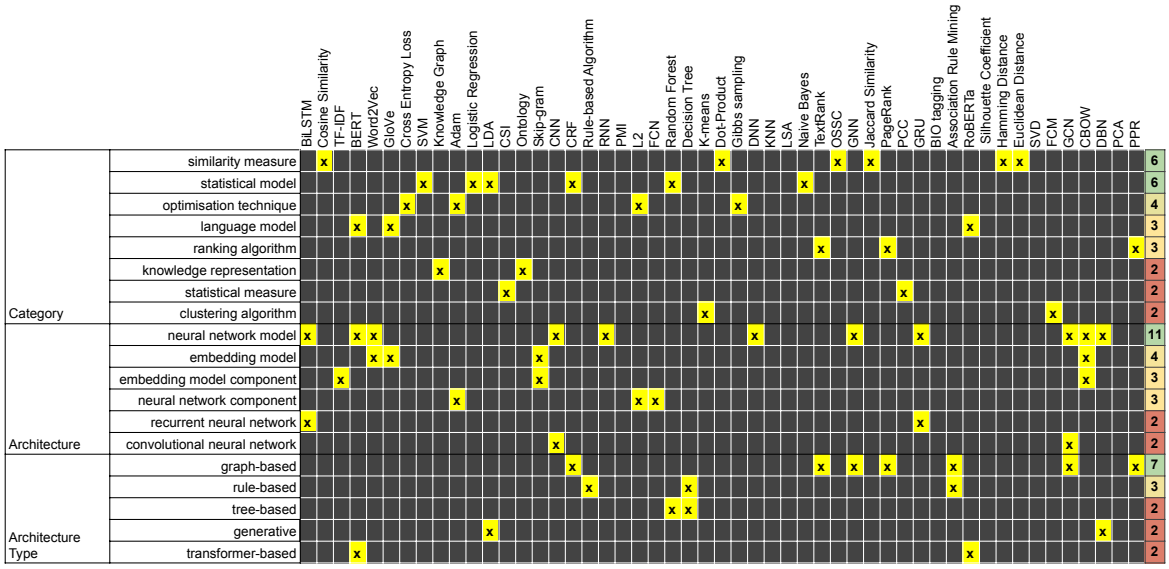

Figure 3.8: Categorization of most common Models, Techniques and Algorithms

3.7.1 Machine Learning Applications

As described in Section 3.5.4, the retrieved administrative data provides a thorough overview of machine learning applications in many domains and tasks. While these scholarly publications did not just concentrate on software feature extraction, they were chosen based on a specific set of relevant keywords related to this domain. Hence, the data is a priceless asset for finding common machine-learning tasks and models that might be adjusted for software feature extraction. This section thoroughly examines the data to improve the choice of machine-learning algorithms at various phases of our feature extraction process. Figure 3.9 displays the frequency of the most often used approaches mentioned in Section 3.6.1 together with their broader applications as described in the literature.

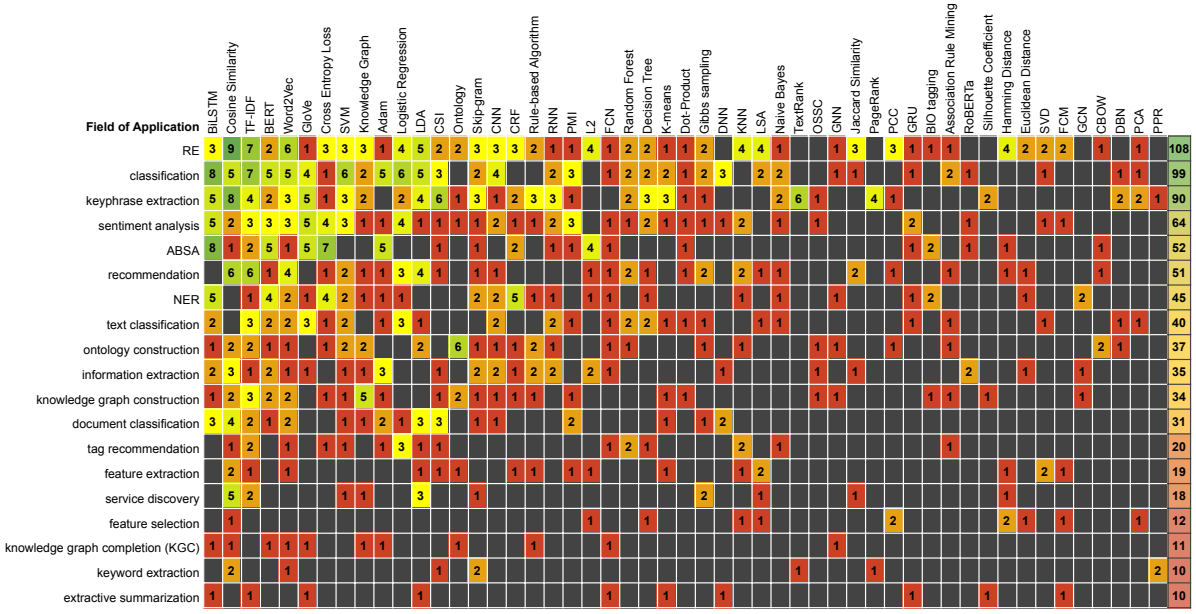

Figure 3.9: Model appearances in literature in conjunction with the broader application fields

The software feature extraction pipeline design involves several machine learning tasks that are of crucial importance. At first, applications that use sequences, such as named entity recognition (NER) and keyphrase extraction, might play a crucial role in organising the raw text or identifying the most

important parts of the text for extracting detailed features. Document classification is essential for effectively organising and categorising the vast amount of data collected. Deep learning systems, such as Aspect-Based Sentiment Analysis (ABSA) (Hua et al., 2024), can provide contextual understanding by identifying the sentiment toward particular features or software qualities, such as maintainability and portability, as outlined in ISO 25010. A similar task is undertaken in (Liang et al., 2023), where the authors use TextRank to extract keywords related to software quality attributes. Methods for constructing a knowledge graph might provide insights into how to organize the retrieved characteristics. Incorporating these activities strategically into the design might lead to a stronger and more flexible artifact.

3.7.2 Pertinent Application Fields

Although the literature study covers several application fields, some domains are particularly relevant to the automated extraction of software features. Through the analysis of various areas, some models, methods, or procedures that have not been regularly observed in the entire data may arise as potentially valuable.

Service Discovery

The unsupervised method of using deep variational autoencoders (VAEs) (Kingma & Welling, 2019) for service discovery, outlined in (Lizarralde et al., 2020), could be particularly useful in the feature extraction phase. Specifically, autoencoders could encode complex software features into a lower-dimensional latent space, making it easier to identify and categorize features from vendor websites automatically. Xin et al. (2021) employ knowledge graph embeddings and estimates the relevance between Mashup requirements and existing services to generate an API recommendation list. N. Zhang et al. (2019) discuss an approach to RESTful service discovery that leverages unsupervised learning techniques like K-Means and topic models to group similar services based on textual features, which could be incorporated into the feature extraction pipeline.

Tag Recommendation

Izadi et al. (2020) employ a multi-label classification approach to recommend tags for GitHub repositories, where the input data consists of textual descriptions and README files. The paper treats the problem of assigning existing topics to new repositories as a multi-label classification problem, which could be adapted to assign multiple software features to a given software product. In the scenario where only a small labeled dataset is available, a semi-supervised approach could be adopted by using the labeled data to train an initial model and then applying it to the unlabeled data to make preliminary feature assignments. These preliminary assignments could then be used to train the model further, iteratively refining its ability to identify software features.

In the same manner, the authors of (J. Zhou et al., 2019) employ a semi-supervised method to address the issue of recommending tags for Docker repositories. They accomplish this by incrementally creating training data and expanding the vocabulary. More precisely, it utilises a logistic regression (LR) model trained on manually annotated training data and a labeled LDA (L-LDA) model as the predictor, using a tagged Docker repository collection. The modified L-LDA enables the integration of labelled data into the topic modelling procedure, hence enabling the model to be directed towards specific software features. In the case of recommending labels for GitHub problems, J. Wang et al. (2021) utilize pre-trained contextual language representations, such as BERT, to tackle the problem of polysemy by employing contextual embeddings. These embeddings can more accurately represent words based on the context in which they appear, thereby effectively addressing the problem of polysemy. This is especially applicable to the process of extracting software features, as a feature may be stated in several ways across different texts.

3.8 Learning Types

In the next segment of the analysis of the same data (section 3.5.4), the focus is redirected towards the types of learning—unsupervised, semi-supervised, and supervised—utilized in the academic articles from which the models were obtained. This aspect of the data is vital as it uncovers the learning environments in which these models are most commonly employed. These insights might be crucial in making well-informed decisions for the design of the feature extraction tool, especially considering the project's emphasis on unsupervised learning with the potential for including semi-supervised approaches.

Figure 3.10 indicates a strong inclination towards certain models in unsupervised learning scenarios. Cosine Similarity and TF-IDF emerge as robust choices for tasks such as similarity measurements and information retrieval. LDA shows promise for topic modeling and organization of extracted features. Interestingly, Knowledge Graphs and CSI also appear to be significant in unsupervised contexts, suggesting their potential for complex relationship mapping and feature extraction. While the project focuses primarily on unsupervised learning, the prevalence of language models in supervised contexts raises an interesting possibility. These could be integrated semi-supervised, working in tandem with unsupervised techniques to enrich the feature extraction process.

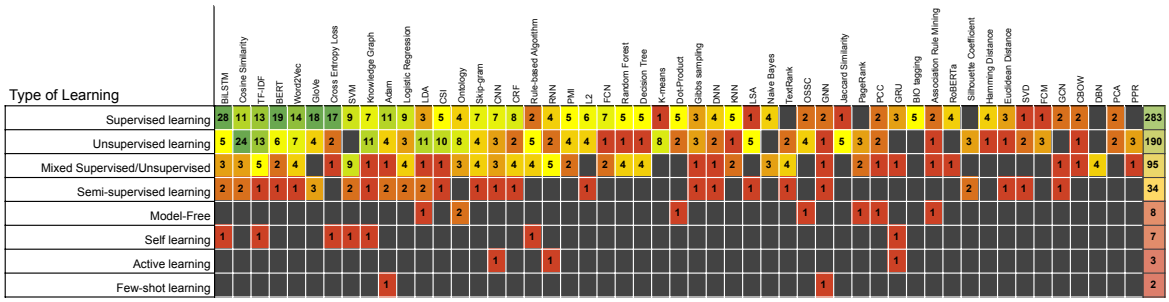

Figure 3.10: Frequency of model occurrences across different learning contexts

3.8.1 Implementation Features

The next analysis shifts its attention from machine learning models to thoroughly examining the features and functionalities that often come with these models. The data presented in Figure 3.11 has two main objectives: firstly, to clarify the different elements and layers that contribute to the complexity and effectiveness of machine learning implementations, and secondly, to provide an organised framework for comparing a wide range of techniques and methods. The knowledge derived from this data can guide the design choices for the artifact, encompassing data representation, pre-processing, learning methods, and specialised techniques.

The universality of features such as text embeddings and tagging highlights the need for robust data representation and initial text pre-processing operations. Similarly, the importance of POS tagging and removing stop words emphasises the need for linguistic refinement in order to prepare the data for further analysis. The emergence of named entity recognition as a significant feature suggests the possibility of a more focused approach to extracting features and mapping relationships. The inclusion of topic identification and clustering characteristics provides the opportunity to categorise comparable software features, potentially streamlining the decision-making processes related to the artefact. The use of seed words in domain-specific methodologies offers a distinct prospect. By utilising pre-existing software features as initial inputs, the artifact can facilitate the extraction process, correlating with previously discussed topic discovery techniques such as labeled LDA. Cosine Similarity, BiLSTM, and BERT are widely recognised and commonly used models that demonstrate versatility and effectiveness in several aspects of artifact design, making them universal options. Additional prominent characteristics, such as the representation of data in a graph format and using measures to determine similarity, provide various possibilities for designing the artifact. These possibilities include using graph-based learning techniques or including recommendation capabilities.

Figure 3.11: Mapping of features to models in the literature. The table presents the comprehensive mapping of 60 distinct features to the corresponding models, grouped into 13 categories. For detailed definitions and explanations of the features, please refer to Appendix B.

Overall, the data presented above offers indications about various aspects of the implementation that should be considered. These include data pre-processing, input representation, different machine learning techniques that can be used, potential output formats, domain-specific techniques, and optimisations.

3.8.2 Tools/ Libraries/ Resources

The literature highlights a wide range of tools, libraries, and resources, as seen in Figure 3.13a. These can be roughly classified into several classes according to their main functionality. CoreNLP, NLTK, and

SpaCy are renowned for their extensive text processing and natural language comprehension capabilities. They are particularly valuable for making initial data modifications and extracting features. Tools such as Gensim and fastText and libraries like Sci-kit Learn are specifically designed for machine learning tasks and may be highly useful for constructing and assessing models. WordNet, DBpedia, and Wikidata are useful information bases that might assist with semantic analysis and contextual comprehension. The presence of ProgrammableWeb, despite its deprecation, indicates the necessity for a thorough software repository to assist in verifying or enhancing the feature extraction procedure. Investigating other options instead of ProgrammableWeb might find similar advantages, resulting in a more comprehensive collection of data sources for the artifact design. The abundance of these resources in the literature underscores their dependability and effectiveness. The usage details of these tools, libraries, and libraries may be found in Appendix D.

3.8.3 Datasets

The collected data on datasets and their yearly uses, presented in Figure 3.12 elucidates evolving research interests and resource preferences in the field. Based on the yearly totals for each dataset, it appears that their usage has been relatively consistent over the years, ranging from 2016 to 2023, without any drastic changes.

	SemEval	20 Newsgroups	Yelp	Reuters-21,578	Inspec	DUC	Amazon reviews	IMDB Reviews	ProgrammableWeb	BBC Sport / BBC	OHSUMED	Amazon Electronics Reviews	OHSUMED	PubMed-10K	TREC	Krapivin2009
2023	1	1	1	1	1	1		1		1	1		1			
2022	9		2						1			1				2
2021	5	3	2	3	1	1	2	1	1		1		1		2	
2020	5	4	3	2	2	2	2		1	1	1	1	1	3		
2019	6	2	1	1	2	1	2		2	1						
2018	1	2	1	2				3		1	1	1	1		1	
2017			1			1						1				
2016																
	27	12	11	9	6	6	6	5	5	4	4	4	4	3	3	2

	RCV1-V2	Amazon Books Reviews	Genia	Stanford Sentiment Treebank	NYT	FB15K	CoNLL	PUblic Requirements (PURE)	CrossNER	Nguyen2007	Theses100	wiki20	Github Repo	Spambase	Ionosphere	PROMISE “NFR dataset.”
2023					2			1								
2022										1	1	1				
2021	1	1							1				1	1	1	
2020	1					1	1									1
2019			1			1	1									
2018		1	1	2												
2017																
2016																
	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

	GitHub repositories	DBLP	Cora	CiteSeer	JRC-ACQUIS	Enron	Multi-Domain Sentiment Dataset	Stack Overflow	Multi-Level Event Extraction (MLEE)	ACL	Ontology Alignment Evaluation Initiative (OAEI)	Large Movie Review dataset	Structured Web Data Extraction Dataset (SWDE)	Common Crawl	WebNLG	FewRel
2023		1	1	1							1				1	
2022																
2021					1		1	1								
2020						1										1
2019	1															
2018									1	1		1	1	1		
2017																
2016																
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	WNut_17	Book Crossing	MovieLens	Microsoft Research Paraphrase Corpus (MRPC)	App Reviews / Descriptions	Sentiment140	STS-Gold	WN18	DBpedia	MAG	Github projects	Senseval	MUC	T-Rex	OntoNotes	
2023					1											19
2022		1	1								1					21
2021	1											1			1	35
2020						1	1	1					1	1		39
2019				1					1	1						25
2018																23
2017																3
2016																0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

URL /Ke ase ase com /Ke /dat sd.e 25nm /o /da .co/ sau efar xtra edu xtra /dat oha proje and ole atas ree/ org/ de.c xtra xtra xtra b/e edu/ edu/ NoR N/A atas dul etw lang ts/v anc own k/M uag hin /~a 576 mor ole e.cc .co s/so s/pr data m/js tase sets ocs. ib.co seag ithu /dat eco hub. itaset

Figure 3.12: Datasets commonly used in the literature. The table includes the names of the datasets and their corresponding URLs.

The persistent utilisation of datasets over time indicates a steady interest in well-established NLP and machine learning tasks. Significantly, datasets such as ‘SemEval,’ ‘20 Newsgroups,’ and ‘Yelp’ are particularly prominent in the literature due to their frequent usage. An example of a dataset often employed for sentiment analysis and other semantic tasks is the ’SemEval’ dataset. This dataset might offer instructions on how to efficiently extract and analyse contextual information from textual input. The ’20 Newsgroups’ dataset, frequently employed in text classification tasks, has the potential to provide valuable insights for the organisation and categorization of extensive amounts of unstructured textual data. The ‘Yelp’ dataset, mainly used for review analysis, might demonstrate the significance of user-generated material as a data source for extracting features. The existence of information retrieval datasets implies a requirement for sophisticated data querying and ranking capabilities inside the artifact. In addition, it is worth noting that datasets such as ‘ProgrammableWeb,’ ‘GitHub Repositories,’ and ‘Stack Overflow’ may also be highly pertinent to the current work while being less commonly referenced. For example, the website ’ProgrammableWeb’ may provide information about various software services and APIs, ’GitHub Repositories’ could offer practical illustrations of software feature documentation, and ’Stack Overflow’ could provide user-generated material related to software development problems.

3.8.4 Evaluation Methods

Within the complex realm of machine learning research, evaluation metrics play a crucial role in verifying and evaluating the performance of models. The literature study discovered a diverse range of measures,

the frequency of which is presented in Figure 3.13b. Recall and precision, frequently employed together, are the most widely used metrics, with each of them appearing 107 times. The F1 Score closely trails after, being referenced 102 times. This makes sense, considering that the F1 Score combines both precision and recall into a single metric, providing a balance between the two. The frequent co-occurrence of these three measures suggests their substantial impact on tasks like classification, keyphrase extraction, and named entity recognition (NER), which are commonly studied areas of research. Classification, in particular, appears to be a very significant task handled in 28 studies. Hit Ratio (HT) appears eight times in the literature and is commonly associated with recommendation systems. This statistic quantifies the frequency at which recommended items are deemed relevant or 'hits,' providing a gauge of the recommendation quality of a model. Since the suggested system may potentially be involved in software recommendation, the Hit Ratio might be a suitable measure for analysing this aspect.

	Tools/ Libraries/ Resources
33	CoreNLP Tool
22	WordNet
19	NLTK
8	SpaCy
6	Gensim
5	fastText
5	Sci-kit Learn
5	DBpedia
5	ProgrammableWeb
5	Freebase
5	Wikidata
4	SentiWordNet
3	Bing Liu Opinion Lexicon
3	Stanford OpenIE
2	SenticNet
2	BabelNet
2	Probase Knowledge Base

(a) Most utilised Tools, Libraries and Resources

	Evaluation Measures
107	Recall
107	Precision
102	F1 Score
43	Accuracy
14	Ablation Analysis
9	MAP
8	HT
8	NDCG
4	AUC
4	t-test
3	Chi-Square
3	ROC
3	ANOVA
2	Rouge-L
2	LRAP
2	MSE
2	RMSE
2	Perplexity

(b) Evaluation methods commonly used in the literature

It is worth noting that the metrics discussed in the literature frequently appear in many forms. For example, metrics such as precision, recall, and hit ratio can be computed using different approaches, including micro, macro, and weighted averaging. They can also be defined at a certain cutoff point, such as @N in recommendation tasks

- *Micro-Averaging*: This approach combines all instances of true positives, false positives, and false negatives from several classes to calculate performance measures such as Precision and Recall. This is especially advantageous when there is a class imbalance.
- *Macro-Averaging*: In this case, the metric is computed separately for each class, and subsequently, the results are averaged. This approach ensures equal treatment of all classes, irrespective of their sizes, and is well-suited for multi-class situations where the performance of each class has significance.
- *Weighted Averaging*: This approach is a middle ground between micro and macro averaging, where the metric for each individual class is calculated by taking the average, with the weight of each class determined by the number of samples it contains. This is applicable in multi-class classification problems with a somewhat skewed class distribution.
- *Hit Ratio @N*: Frequently employed in recommendation tasks, this metric quantifies the frequency

with which a recommended item is included in the top-N list of a user's true preferences. This is especially advantageous when suggestions must be constrained to a specific quantity of items.

- *Precision@N and Recall@N*: In certain situations, such as recommendation systems or search engines, it is often more important to assess the quality and relevance of the top N outcomes rather than examining all potential classifications or retrievals. Precision@N and Recall@N offer a targeted assessment of the system's performance at the highest positions in the list, which is typically of utmost importance to users.

Specialized Metrics:

- *Area Under the Curve (AUC)*: This metric evaluates the model's ability to distinguish between classes and is generally used in binary classification tasks.
- *Normalized Discounted Cumulative Gain (NDCG)*: Often used in ranking and recommendation systems, NDCG measures the quality of the ordered list of recommendations.

Ablation Analysis, referenced in 14 articles, is a specialised assessment approach employed to evaluate the influence of various components on the overall performance of a model. It entails the methodical elimination of various elements to assess their impact on the ultimate result. Due to the modular structure of the proposed feature extraction artifact, Ablation Analysis can provide useful insights into the components that are most essential for efficient feature extraction.

3.8.5 Evaluation Strategies for the Feature Extraction Artifact

Given the unsupervised nature of the proposed feature extraction system, conventional supervised evaluation metrics like Precision, Recall, and F1 Score may not be directly applicable. Nevertheless, using a two-phase training strategy might be a feasible resolution:
Initial Supervised Training: Available datasets could be used for initial supervised training, focusing on the multi-label classification of software features. In this phase, conventional metrics like Precision, Recall, and F1 Score could serve as effective evaluation measures.
Unsupervised Phase: The second phase could leverage unsupervised methods to refine further and extend the list of software features, guided by the initial model's confidence scores and predictions. Alternative evaluation metrics could include:

- *Topic Coherence*: To assess the quality and relevance of the generated software features.
- *Silhouette Score*: Useful for evaluating the quality of clustering of similar features.

3.8.6 Decision Making Process

A systematic approach to model selection is vital to ensure the most suitable models are used. To guide this process, a decision meta-model is adopted which is situated within the multiple-criteria decision-making (MCDM) field, based on (Farshidi et al., 2024). The decision model was adapted as depicted in Figure 3.14 and offered a structured methodology for identifying the most appropriate combination of models for software feature extraction. There are five steps for selecting a combination of models:

1. **Models**: An exhaustive list of potential models is compiled based on existing literature and technological considerations. Appendix A can be used to understand the definitions of models.
2. **Feature Requirements Elicitation**: Essential features for software feature extraction are identified, taking into account the specific requirements of the task. Appendix B can be used to understand the definitions of features and model characteristics.
3. **Finding Feasible Solutions**: A list of feasible combinations of models and features is generated based on the identified requirements. Figure 3.11 can be used to determine which models support specific features.
4. **Selecting Feasible Combinations**: Suitable combinations are selected from the list of feasible solutions, considering factors like performance, complexity, and alignment with the task. Figure 3.5 provides information on the combinability of models based on the reviewed articles.

5. **Performance Analysis**: Upon determining a set of possible combinations, suitable evaluation measures should be chosen to analyze their performance and validate their efficacy in software feature extraction. Figure 3.13b revealed the most common evaluation measures, while their definitions can be found in Appendix C.

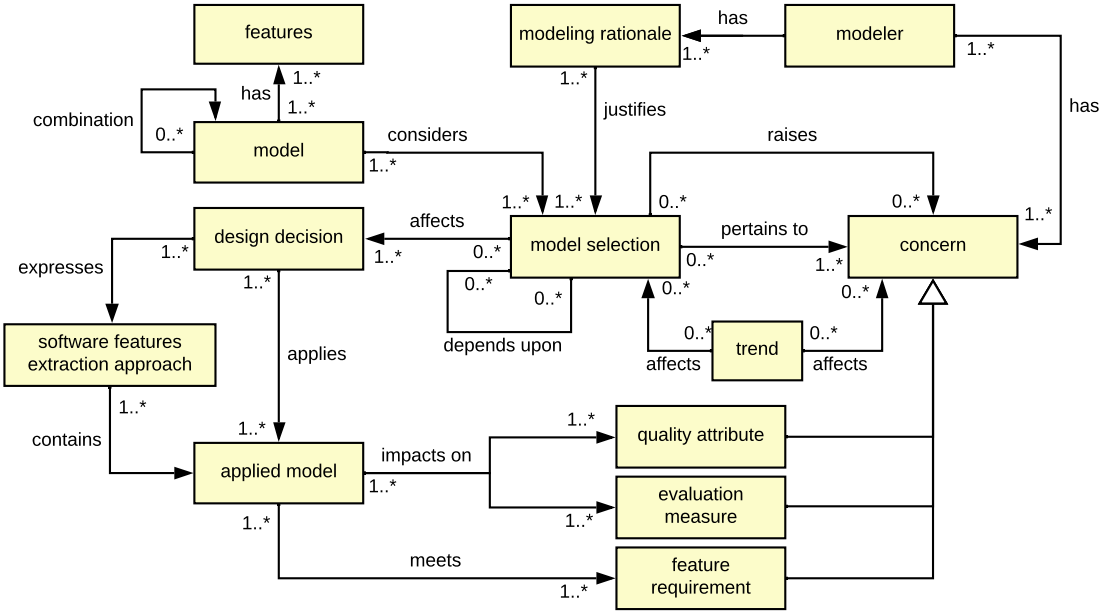

Figure 3.14: The decision-making process employed in selecting software features extraction approaches within the academic literature.

The decision-making process is based on the concepts outlined in the ISO/IEC/IEEE standard 42010[10], which offers a framework for conceptual modelling. Initially, research modellers establish the objectives and considerations for the task, such as extracting software features. Subsequently, the process involves identifying potential models and features that align with these objectives. After selecting a group of models, a thorough study is carried out to evaluate each model's suitability, advantages, and drawbacks. The study considers several criteria, such as the complexity of the model, its interpretability, and the practicality of integrating multiple models. This systematic procedure guarantees the efficient identification of models that meet the precise requirements of software feature extraction. The decision meta-model consists of the following components:

- **Models**: These are the various machine learning models, algorithms, or techniques under consideration.
- **Features**: These are the attributes or functionalities that the models can support.
- **Measures**: These are the evaluation metrics used to assess the performance of the models.
- **Qualities**: These refer to the quality attributes of the models, such as interpretability or scalability.
- **Concerns**: The goals and objectives of the research, as well as any constraints. Features, Measures, and Qualities constitute the concerns.
- **Solutions**: These are the optimal combinations of models that satisfy concerns.
- The formal expression of MCDM in this context can be represented as
  MCDM: Models $\times$ Features $\times$ Concerns $\rightarrow$ Solutions.

[10] https://www.iso.org/standard/74393.html

With the adoption of this MCDM-based decision-making process, this research aims to have a methodical and transparent approach to the selection of models, with the end goal of ensuring the development of a software feature extraction solution that is both robust and effective.

4 Summary of Literature Review

4.1 General Observations

In the systematic review conducted for this thesis, it was noted that only 25 out of the 155 (16.1%) surveyed papers made their code repositories publicly available. This lack of openness poses challenges for replicating studies and advancing the field of automatic software feature extraction. Open-source code is crucial for transparency and reproducibility in machine learning research (Haefliger et al., 2008).

Out of 207 models identified, a significant portion (122, 58.9%) were singletons, limiting the generalizability and comparability of results. For the advancement of unsupervised NLP techniques in software feature extraction and other domains, there is a need for adopting common models or evaluation standards, in order to be able to compare approaches more effectively (Amershi et al., 2019).

Additionally, some papers lacked clear explanations of how multiple models were integrated, making it difficult to understand the methodology and evaluate its effectiveness. Detailed descriptions of model combinations are essential for transparency and for extending the research further (Amershi et al., 2019).

The review also revealed a wide range of model variations, complicating comparisons and replication efforts. The examination of the data showed a considerable variety in the models used, such as RoBERTa, DistilBERT, DeBERTa, DistilRoBERTa, and SpanBERT. All of these are extensions or variations of the well-known BERT language model (Devlin et al., 2019), which itself was featured in 32 of the surveyed papers. A standardized categorization of these models could facilitate their reuse and understanding of the different variations, thereby enhancing collaboration and progress in the field (Sarker, 2021).

BiLSTM was found to be a prevalent model, evidenced in Figure 3.5. The evolution of textual or sequence processing machine learning models has seen a progression from traditional methods like rule-based systems to statistical models, then to recurrent neural networks (RNNs) like LSTMs, and ultimately to transformer-based models like BERT, which have achieved state-of-the-art results across various NLP tasks due to their ability to capture deep contextual information. While traditional models offer ease of use and interpretability, modern models like BERT provide high performance across various tasks, thanks to advancements in computational resources and larger datasets, and are therefore preferred (Ribeiro et al., 2016). However, the more modern approaches introduce increased complexity and a trade-off between interpretability and performance. Analyzing the models and techniques used in similar constituent tasks can direct the design process for the software feature extraction task.

Regarding datasets, less than two-thirds (97 out of 155) of the papers used publicly available datasets, and a large number of these (41) were not reused in subsequent research. This lack of dataset sharing and reuse hampers progress and raises questions about the generalizability and credibility of the research. Encouraging open sharing of datasets could accelerate research and ensure more robust and credible methods in the field.

The scarcity of accessible past datasets poses difficulties for researchers in verifying and replicating published findings. This limitation hinders the objective assessment and comparison of various models, obstructing the recognition of state-of-the-art methods and potential improvements (Pujol et al., 2020). Additionally, the lack of diverse and publicly available datasets could lead to the development and evaluation of models that are biased, thereby restricting their applicability in real-world settings and among varied user groups (Bagdasaryan et al., 2019). This problem is further exacerbated by the redundant efforts in gathering and preparing new datasets, which not only consumes precious resources but also slows down research. To address these issues, it is crucial to encourage a more open and collaborative environment within the research community.

4.2 Threats to Validity

Given the empirical nature of this study, addressing threats to validity is essential for ensuring the integrity of the research findings. X. Zhou et al. (2016) extensively examined standard practices related to Threats to Validity (TTVs) across 316 Systematic Literature Reviews (SLRs) within the field of software engineering research. Consequently, they compiled a comprehensive list of common factors that could impact validity, as depicted in Figure 4.1 . In this section, we examine the potential threats to construct, internal, external, and conclusion validity that may affect this study's outcomes.

4.2.1 Construct Validity

In the context of the Systematic Literature Review, construct validity refers to the process of accurately identifying operational measures for the concepts that are being studied. It is possible that the construct

Category	Definition
Construct Validity	Determine appropriate operational metrics for the concepts under investigation.
Internal Validity	Strive to establish a cause-and-effect relationship in which specific conditions are thought to result in other conditions, differentiating them from misleading associations.
External Validity	Specify the scope within which the conclusions of a study can be applied.
Conclusion Validity	Show that the processes involved in a study, such as the data collection method, can be replicated, yielding consistent outcomes.

Figure 4.1: Definitions of validity categories based on (X. Zhou et al., 2016)

validity might be compromised by a number of different reasons. One of these factors is the selection of databases and sources, which may be restrictive if they do not completely cover all of the literature that is pertinent to the analysis. In order to counteract this, the databases and platforms that were utilised for the literature search are outlined in detail in Chapter 3. This ensures that a diverse variety of publications is taken into consideration. Furthermore, the criteria for adding or excluding research throughout each step of the SLR are rigorously specified in corresponding subsections, which reduces the chance of making decisions that are inconsistent with the requirements of the study.

4.2.2 Internal Validity

When it comes to the concept of internal validity, there are a few aspects that need to be carefully considered in order to guarantee that the findings of the study are accurate. The bias in model selection is the first issue to consider. The natural language processing (NLP) models that were selected for this study could be inherently more effective for certain kinds of software features than they are for others. For the purpose of mitigating this issue, a variety of models will be examined in order to guarantee a feature extraction process that is both balanced and complete. The second area of interest is concerned with regard to the sources of data, . Relying only on the websites of vendors or on the reviews of customers may result in a sort of selection bias due to the fact that various sources may place an emphasis on distinct combinations of features. To solve this issue, the research will make use of a wide variety of review sites and vendor websites in order to construct a dataset that is more representative of the whole. The final point to consider is that the operational definition of "software feature" has the potential to be a source of bias, either because it is too inclusive or exclusive. The criteria for what constitutes a software feature have to be explicitly established and implemented in a methodical manner throughout the entirety of the research in order to reduce the likelihood of this threat occurring.

4.2.3 External Validity

External validity refers to the extent to which the findings of a research may be applied or generalised to a larger population or real-world situations. Considering the exclusive focus of this research on the automated extraction of software features, the findings may not have direct relevance to other fields. In order to improve the capacity to apply the findings to a wider context, the study should encompass a wide variety of software categories and utilise models that have broad applicability.

4.2.4 Conclusion Validity

Conclusion validity pertains to the reliability and credibility of the study's results. In order to guarantee reproducibility, it is essential to meticulously record each stage of the approach. The software repository and datasets will be publicly accessible, facilitating independent verification of the findings.

4.3 Ethical Considerations

In addition to the threats to validity, it is crucial to take into account the ethical implications of automatically extracting features from vendor websites. To uphold the ethical integrity of the study, it is imperative to tackle concerns such as data privacy, intellectual property, and any biases in the models. This work attempts to maintain scientific rigour by diligently detecting and mitigating these threats, in order to establish a reliable and effective approach.

5 Implementation

The following sections focus on the implementation and evaluation details of the artifact, based on the principles of design science. In order to make the explanation more cohesive and intuitive, the process will be explained with respect to the chronological order of the steps taken by the final artifact, from its input web data to the output knowledge graph. Existing approaches are explored in section 5.1, to determine the specific design choices for the system pipeline presented in section 5.2. Section 5.3 discusses the data collection and analysis process, Chapter 6 explains the specific architectural decisions concerning the transformation of the collected data into the desired structured and informative form using classification techniques, followed by the experiments that were set up to evaluate the performance of the techniques explored. Chapter 7 delves into the creation of the knowledge graph using the extracted information and identified features. Finally, Chapter 8 discusses the findings of this research and the extent to which the requirements of the proposed artifact are met, its feasibility analysis, and potential enhancements that can be made in the future. The code used to implement the recommended system is made available on Zenodo (Constantinou, 2024). A detailed Bill of Materials (Appendix G) lists all the tools and resources used in this study.

5.1 Existing Approaches

Hofer et al. (2024) provide an overview of the main tasks for Knowledge Graph Construction with a focus on (semi-)automatic and incremental solutions. The main tasks listed are:

- **Data Collection and Preprocessing:** Selection of relevant sources, retrieval and processing of relevant source data.
- **Metadata Management:** Acquisition and management of metadata, for example structural metadata, temporal information, quality reports or process logs.
- **Ontology Management:** Creation and incremental evolution of a KG ontology.
- **Knowledge Extraction (KE):** Derivation of structured information and knowledge from unstructured or semi- structured data, using techniques for named entity recognition, entity linking and relation extraction.
- **Entity Resolution (ER) and Fusion:** Identification of matching entities and their merging within the KG.
- **Quality Assurance (QA):** Possible quality aspects, their identification, and repair strategies of data quality problems in the KG.
- **Knowledge Completion:** Extending a given KG, by learning missing type information, predicting new relations, enhancing domain-specific data, etc.

These steps are also presented in a generic pipeline in Figure 5.1. The construction pipeline does not need to follow a strict order for each task, and certain steps may be optional depending on the specific use case for the knowledge graph. These are generally the steps followed to create the type of knowledge graphs typically presented in literature, therefore there needs to be some adjustment to tailor the design requirements for this specific project.

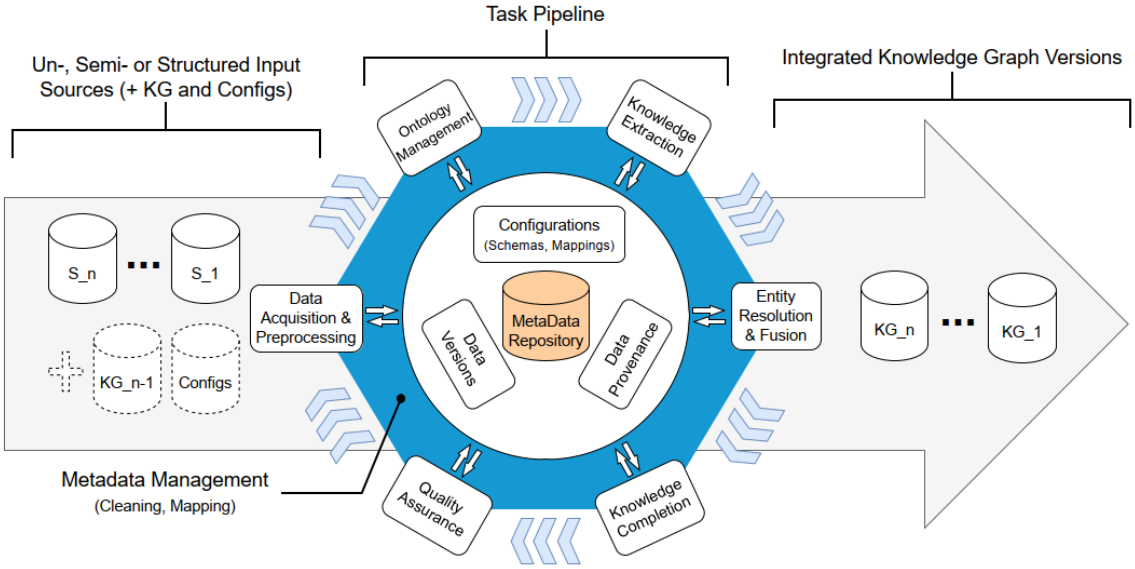

Figure 5.1: Generic pipeline for incremental Knowledge Graph Construction (Hofer et al., 2024)

Generally, the design of KG construction frameworks can differ based on the type of knowledge being processed and the intended application, however, there are two main approaches to developing knowledge graphs: the top-down approach, which emphasizes knowledge schemas like domain ontologies, and the bottom-up approach, which centers on knowledge instances, such as Linked Open Data (LOD) datasets (Z. Zhao et al., 2018). In the top-down approach, which prioritizes well-defined domain ontologies, the ontologies and their schema are established first, followed by the addition of knowledge instances into the knowledge base. In contrast, the bottom-up approach focuses on extracting knowledge instances directly from knowledge resources. In our case, there is not an existing well-defined and exhaustive domain ontology, since the purpose is to create one consisting of newly identified software features. Hence, an incremental bottom-up approach must be taken, which requires the identification of specific knowledge instances in the form of software features, which may then be grouped together in a hierarchical structure. L. Wang et al. (2023) conducted a systematic literature review on the application of knowledge graphs specifically in the software engineering domain. Their findings can be summarized in the table below:

Author	**Technology**	**Advantages**	**Disadvantages**
Wenpeng et al. (2017)	Software knowledge extraction, Entity association extraction	Established linkages between software knowledge entities in different types of software resources	The content update mechanism of the chart needs to be considered
F. Wang (2019)	Identification and extraction of entities, concepts, attributes, and interrelationships	A degree of depth and completeness, a high degree of accuracy, and a rigorous and rich data model	Named entity recognition model still needs improvement
C. Zhou et al. (2018)	Open-source software bug identification methods	Highly capable of feature fusion	Long training hours
H. Li et al. (2018)	Co-citation resolution techniques and heuristics	Avoid software warning messages and be very efficient	Very specialized background knowledge required
Lemos et al. (2014)	Self-constructed knowledge graphs of code term associations	Traceability links between vulnerabilities and software components can be made more accurate	Small data or low-dimensional data may not be able to construct the desired mapping effect

Qin & Chow (2019)	Using techniques such as NLP to add extracted entities as new knowledge to the vulnerability knowledge graph	Automated research into vulnerabilities, with the ability to perform vulnerability tracking	Candidate vulnerability information CVE Chain requires manual tagging and is subjective in nature
D. Du et al. (2018)	A random forest algorithm was introduced to connect the CVE version of the project with the Maven project version by a matching method	The method works well and allows a more accurate tracing of the relationship between the vulnerability and the software component	The ontology matching approach for the three types of resources is analyzed in isolation
J. Zhang et al. (2020)	Capturing the semantic information of bug reports using LSTM, Using CNNs to capture the semantic information of the code	Better capture of keyword information in source files and bug reports	Handling of code and reports in terms of semantics only, relying on parameter definitions

The main areas for which knowledge graph construction is explored in this domain are:

- Network security analysis
- Development vulnerability mining
- Cloud products recommendation
- Bug locating and fixing
- API recommendation

In terms of data collection, researchers primarily focus on software repositories, version control systems, API components, software vulnerability databases such as CWE and CVE, software non-functional requirements, and various description documents. In this project, data will be collected in the form of unstructured text from vendor websites. Some studies (F. Wang, 2019) construct the knowledge representation model or ontology using software engineering experts, and then knowledge extraction of the data is conducted according to the model or ontology.
Knowledge extraction typically consists of entity extraction, attribute extraction and relation extraction (Z. Zhao et al., 2018). In existing research, entities and their associations are primarily identified using rule-based, machine-learning-based, and deep learning-based methods (L. Wang et al., 2023). Some researchers suggest software-specific NER techniques (Feng et al., 2018) while others employ conditional random field (CRF) and bi-directional long and short-term memory (BiLSTM) neural networks for named entity recognition (Nayak et al., 2020). However, this project primarily focuses on software feature extraction. Even though some named entities may also be included in the intended knowledge graph, such as specific product support (e.g. VMWare), the categories of software features that may come up do not necessarily need to refer to specific entities, therefore a different approach was chosen to locate these features from the unstructured text data. Specifically, a custom noun phrase extraction (section 5.5) method will be used, which is expected to also detect frequently mentioned entities. Once these are extracted, the construction of the knowledge graph boils down to classifying which of these phrases represent pertinent software features. Then, as discussed in section 5.4.2, entity extraction and linking will be handled indirectly by incorporating data from an existing, open-world knowledge graph, namely DBPedia[11], and associating the identified features with nodes of that knowledge graph. The main challenge of existing approaches is their inherent need for large amounts of labeled data or manual curation of rules for knowledge extraction, or labeled data for the classification of which entities to include in the knowledge graph. This project attempts to mitigate these problems by utilizing recent techniques for the classification of features and their associations. The pipeline used to generate the knowledge graph is provided in the next section, followed by detailed explanations for each of its parts.

[11] https://www.dbpedia.org/

5.2 Knowledge Graph Generation Pipeline

The following diagram presents the proposed system pipeline, split into two main parts. The first part is responsible for collecting the required textual data from the cloud vendor's web sources, as well as domain-relevant information, to aid the second part in converting the collected data into the intended structured form: a knowledge graph. Subsequent sections provide detailed explanations of each component or process involved in the pipeline concerning the order of execution.

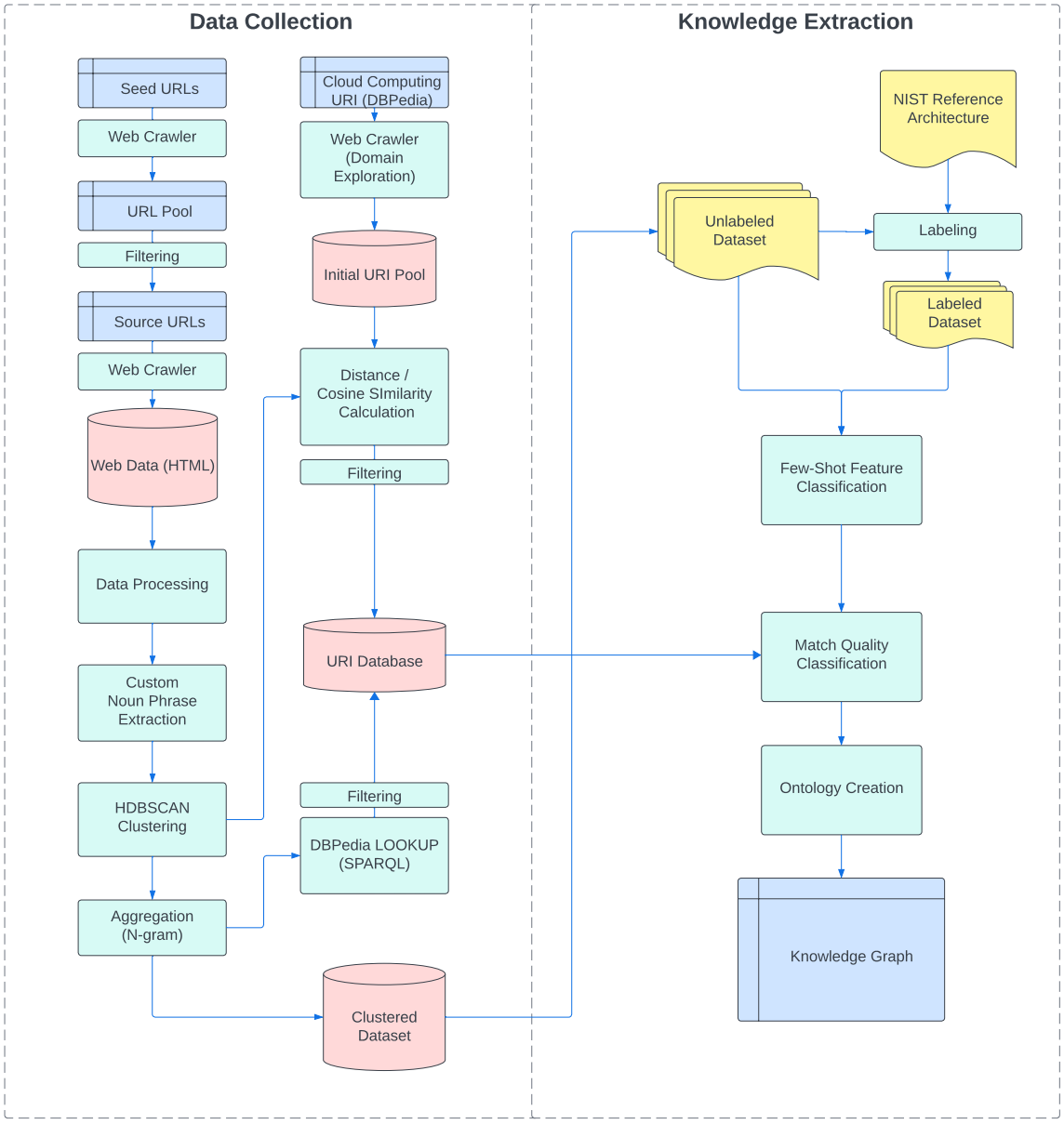

Figure 5.2: The proposed system pipeline for the creation of the knowledge graph.

5.3 Data Collection and Processing

The first step towards building an effective knowledge graph is to find the right data sources. The collected raw text data make up the raw material that constitutes the building blocks of the output graph, therefore the sources of this data should be chosen carefully.

5.3.1 Domain Selection

Originally, this project concerned general software feature extraction and subsequent knowledge extraction by organizing features hierarchically. However, it is generally a good approach to simplify a problem into smaller, more manageable tasks to test its efficacy. In this specific scenario, narrowing down the domain of the problem may also have practical benefits, considering the following assumptions:

- The enormous diversity of software products
- The variability in software descriptions across vendors
- The complexity of categorizing features into a universally applicable knowledge graph

Therefore, the potentially high entropy in data if software features from various domains and categories were explored led to the decision to narrow the focus to cloud vendors and the services they offer in hopes of a more reliable pattern detection and feature classification.

Moreover, if we consider the collected data a snapshot of what is currently advertised, a huge amount of data would be required to adequately represent the features of any kind of software, which would render its further processing and classification impractical and computationally intensive. The assumption here is that if the system performs well for a specific domain, the same pipeline could be reused for different domains as well.

5.3.2 Data Sources

Thirty vendors were selected (30 in total) from whose websites the data was collected. This set includes well-known corporations that are not exclusively active in the cloud service market, such as Google or Amazon, and smaller companies that specialize solely in cloud computing.

The data collection process is initialized using a seed URL for each vendor, which was selected to be the home page of each vendor's official website or the initial page concerned with cloud computing for vendors who also offer different services.

The python library *Selenium* was used to extract the HTML content from each seed URL due to its multiple browser supportability and the better handling of websites that utilize JavaScript to render their content, especially when cookies handling interfaces are present. This is also the case with any HTML rendering step involved in the data collection process. The *robots.txt* file was retrieved for each website to adhere with their data collection guidelines.

From the extracted HTML content, all the links to other web pages that were present were located and stored. A filtering function is applied to the retrieved URLs to ensure their relevance to the domain of interest and exclude pages whose content may not be in the conventional prose text form (e.g., code snippets). This filtering process involves parsing the path components of each URL (as shown in Figure 5.3). URLs potentially containing unsuitable content are excluded based on a pre-defined list of terms commonly associated with such content. This list includes terms like "contact", "docs", "resources", "blog", "manual", "tutorial", "case", "community", and "help".

An example of a discarded URL and its corresponding path components is provided to illustrate the filtering process:
URL: *https://www.lumen.com/help/en-us/home.html*
Parts: *"help"*, *"en-us"*, *"home"*
It is very common for vendor websites to include pages with an ancillary role, such as contact form pages, blogs, and support pages, as opposed to product descriptions and advertisements, which are the target

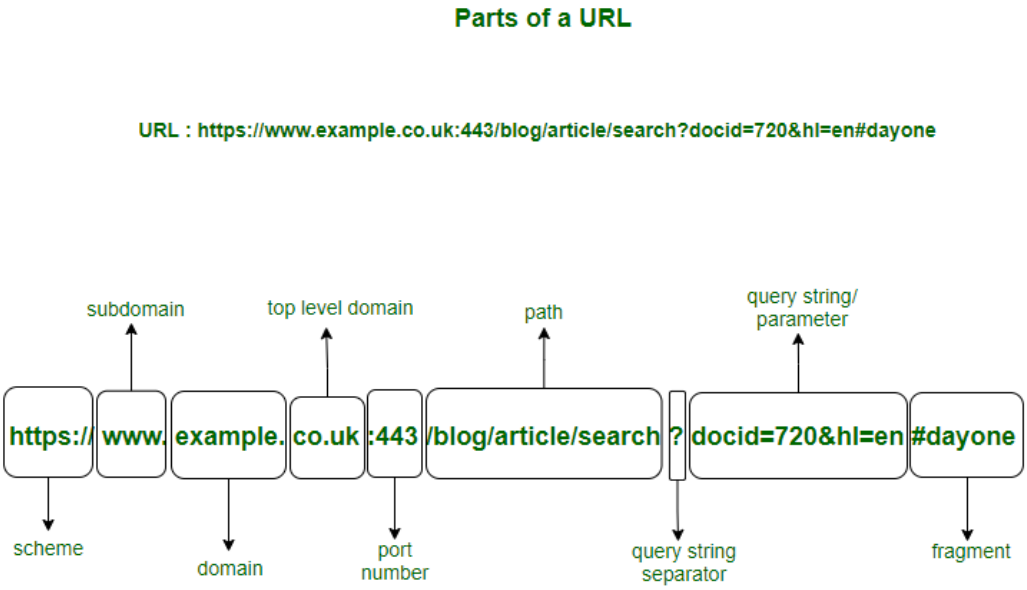

Figure 5.3: URL path parts (source: `https://www.geeksforgeeks.org/components-of-a-url/`)

source of information. Another filter applied was to discard URLs of different domains to the seed URLs to focus on the vendors' commercial websites. The final number of URLs obtained for each vendor is shown in table 2. After the filtering, the final list of the source URLs is determined, from where the input data to the pipeline is found. The HTML content of each web page is retrieved, and then the data is processed.

5.3.3 Data Processing

HTML (Hypertext Markup Language) is used to create web content. HTML documents are text files that use tags to indicate different types of content and formatting. For example, the <p> tag is used for paragraphs, <h1> to <h6> for headers and <a> for hyperlinks. All HTML elements in a webpage have a nested structure that determines its layout. The larger an HTML document is, the more complex the structure becomes, which is often the case with eCommerce sites that include nested menus, product grids, interactive elements, etc. This introduces an enormous variety in the structure of HTML documents across different vendors' websites, which is enhanced by the various design choices of each website's creator. Processing an HMTL document to obtain its main content (in our case, product and service descriptions) is a challenging task and an active research area, as the main content of a webpage is often surrounded by other boilerplate elements related to the template (Alarte & Silva, 2021).

In this project, the same task was approached in a rather simplistic manner: The content was organized into headers for indexing purposes and to guide the content collection process. The header tags are collected and filtered (to exclude cookie content, header menus, footer content, etc.) to focus only on the main content of each webpage, which is more likely to include feature-specific information. Then, for the remaining headers, the text from all the HTML elements that follow the header is extracted as individual 'content items,' where each item corresponds to the text enclosed in a specific HTML tag. These include paragraphs, ordered or unordered lists, table rows, and more. This granular organization of the content into distinct text segments serves an important purpose because specific tags (and content) are often repeated across different web pages. It is common practice in web development to use templates for adjacent linked pages within the same navigational branch of the website, which results in the repetition of content. Content duplication could dilute relevant, feature-specific content and make feature detection harder, which is mitigated by keeping unique copies of each header and its contents.

Table 2 breaks down the list of vendors for which data was collected, the seed URLs to their websites, the total number of relevant URLs extracted, and finally, the total number of unique headers and contents

items collected for each vendor.

Table 2: Vendor Web Sources Breakdown

Vendor	**Source Count**		
	Unique URLs	**Unique Headers**	**Total Contents**
Alibaba Cloud	57	1232	5858
Amazon (AWS)	207	2177	9805
Atlantic.Net	74	1011	4433
Bit Refinery	3	25	65
Brightbox	7	76	331
CenturyLink	154	767	6032
CloudSigma	7	25	247
DigitalOcean	78	1065	3260
Fujitsu	109	1017	4304
GigeNET	17	106	372
Google Cloud	267	3239	19 187
Hyve	128	716	4776
IBM Cloud	15	84	865
Ionos	103	1609	7154
JoyentCloud	9	92	313
Leaseweb	73	668	5774
M5 Internet Hosting	29	183	1361
Microsoft Azure	580	5004	43 680
NTT Communications	119	1596	5407
Neterra	9	87	372
OVH	182	2617	10 875
Oracle Cloud	128	2121	10 701
Rackspace	132	1140	9415
Tilaa VPS	51	450	3312
VPS.NET	15	91	684
e24cloud	35	129	656
CloudScale365	9	157	505
exoscale	58	470	2085
sherWeb	48	446	1917
zettagrid	27	115	380

5.4 Domain Exploration

With the input data now available, the next step is to decide what patterns need to be identified in the data, from which information will be extracted. Familiarizing yourself with common domain terminology would be a reasonable way to grasp an idea of what a cloud service feature is.

5.4.1 NIST Cloud Computing Reference Architecture

An excellent place to start is the NIST Cloud Computing Reference Architecture document[12], which is also used in later stages of the pipeline, to label portions of the data.

The National Institute of Standards and Technology (NIST) is a well-known and established U.S. federal agency that develops technology, metrics, and standards. It is known for its detailed publications that provide clear standards and guidelines across various industries, including information technology and cybersecurity. Therefore, the terminology presented in the document above is also expected to be present in the collected data. The target artifact, or knowledge graph, should also represent the standard knowledge and concepts in the domain of interest to be widely comprehensible and, as a result, useful in practice. The document goes a step further, providing a cloud taxonomy (Figure 5.4) that lays out the central elements of cloud computing in a hierarchical structure, which is closely related to the expected structure of the resultant knowledge graph:

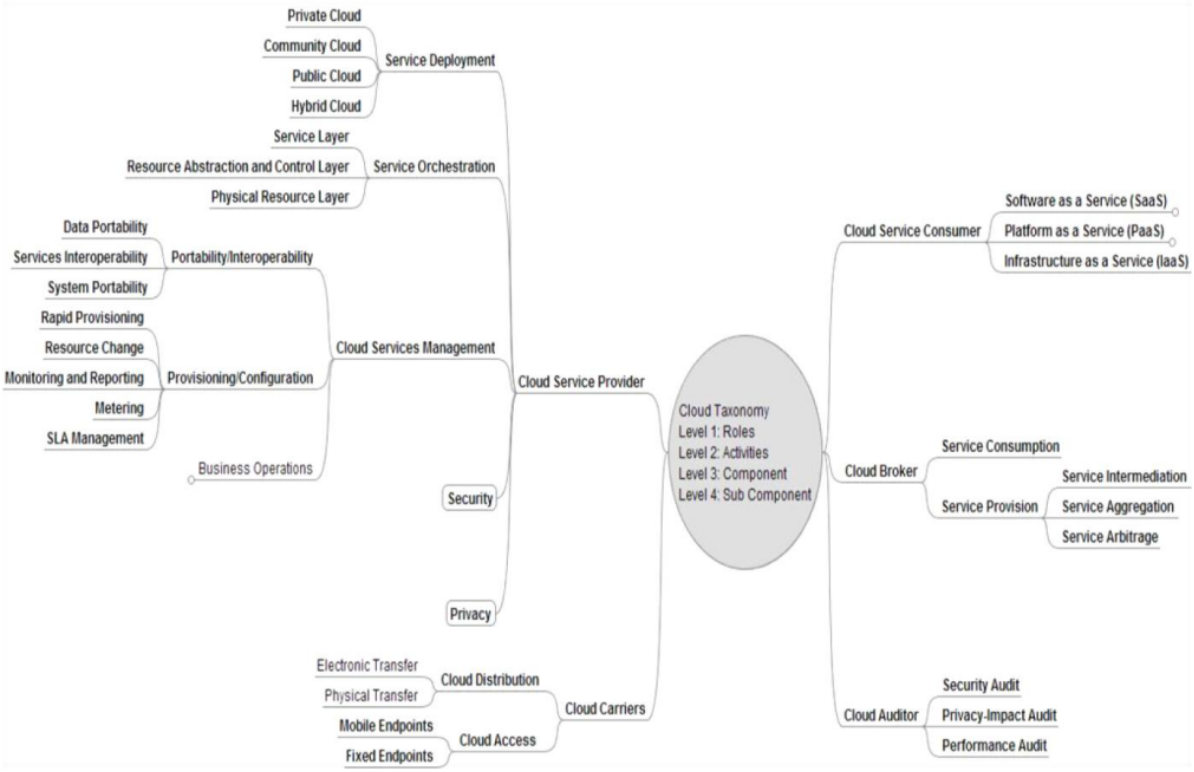

Figure 5.4: NIST Cloud Taxonomy (source: `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-292.pdf`)

5.4.2 DBPedia

DBpedia (“DB” for “database”) aims to extract structured content from the information created in Wikipedia. This structured information is readily available on the World Wide Web and allows users to query relationships and properties of Wikipedia resources semantically. Each Wikipedia resource has its own URI (Uniform Resource Identifier), which is a unique identifier for entities within the DBpedia

[12] `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-292.pdf`

dataset. These relationships and properties are also associated with the structure of a knowledge graph, as it mirrors a real-world ontology, organizing information in triplets of subjects, predicates, and objects. A Resource Description Framework (RDF) structure is fundamental in semantic web technologies. Each triplet forms a statement about a resource:

- **Subject:** Represents the entity/concept/thing for which the described property holds.
- **Predicate:** Describes the relationship or attribute of the subject.
- **Object:** Is the value or another entity that is linked to the subject through the predicate.

The most common relationship between Wikipedia resources is expressed through the *"dbo:wikiPageWikiLink"* predicate and states that the object, a Wikipedia page for a specific term/ entity, is referenced in the Wikipedia page of the subject term/ entity, via a hyperlink. This is a clear indication of the relevance of the two terms, conceptually. Some other examples of predicates are:

- Web browser is the *"Genre Of"* Google Chrome.
- Software is the *"Type Of"* Google Chrome.
- Google is the *"Developer Of"* Google Chrome.
- Google Chrome is the *"Disambiguate Of"* Chrome.

In (Weikum et al., 2021), it is advised to initiate KG construction using extensive, curated "premium" data sources like Wikipedia and other knowledge graphs such as DBpedia. Additional data sources should then be identified and incorporated to capture more specific entities and their relationships. The potential relevance and value of DBPedia as a source of information in this project is also supported by its presence in the analysis of the most common tools, libraries, and resources found in the relevant literature during the systematic literature review phase (3.8.2). The presence of abstracts for the majority of DBPedia resources was also a decisive factor, as it provides textual descriptions of concepts and terminology that may appear in the data but are not always explained in their context, which could limit the ability of the artifact to interpret their relevance and eligibility to be included in the knowledge graph. An example sentence from the data is: "Updated Linux kernels, user-mode software, and toolkits are provided". In this example, "Linux kernel" can be considered as a potential cloud service feature, however, its mere mention in a sentence does not contain information on what it is, which makes its classification challenging. The abstract of the DBPedia page concerning "Linux kernel", states: "The Linux kernel is a free and open-source, monolithic, modular, multitasking, Unix-like operating system kernel". The second use of DBPedia arises from its graphical structure and organization. In addition to the textual description (abstracts), each DBPedia resource page lists relationships with other resource pages in the form of triplets (subject—predicate—object).

The method used to explore the DBPedia resource in this project was to start from the central concept in the domain. Cloud Computing has its resource page: `https://dbpedia.org/resource/Cloud_computing`. and also has a concept page: `https://dbpedia.org/page/Category:Cloud_computing` Concept pages in DBpedia are linked through the *skos:broader* predicate, which portrays a hierarchical relationship between wider and narrower concepts. So starting from the cloud computing concept, narrower and narrower concepts were recursively collected, along with the resource pages linked to each concept, through the *dcterms:subject* predicate (every resource page may or may not have a concept page as a subject). This process resulted in the initial URI pool of DBPedia resources, which will later be matched to the identified features and terminology in the data to derive associations between them and provide textual descriptions to enhance their understanding. The "skos" and "dcterms" parts of the predicates shown above reveal the namespace in which these relationships belong. Namespaces are part of semantic web technologies used in RDF to provide structured ontologies and vocabularies, allowing for more organized and interoperable data across different systems. The namespaces used in this project are:

- **dbo (DBpedia Ontology):** provides structured classifications and properties that describe relationships and attributes of Wikipedia data.

- **dbp (DBpedia Property):** used to assert information about DBpedia resources, often derived directly from Wikipedia infobox data.
- **dbr (DBpedia Resource):** defines resources in DBpedia, where each resource corresponds to a specific Wikipedia page or concept.
- **dcterms (Dublin Core Terms):** provides a broad range of descriptive properties for resources.
- **skos (Simple Knowledge Organization System):** used to build controlled vocabularies like thesauri, classification schemes, and subject heading systems within the RDF framework.

These namespaces can be accessed using the *rdflib* python library. Information about its relationship with other resources and other defined properties can be extracted when given a resource URI. The *SPARQLWrapper* library was also used for more advanced queries for the same purpose.

5.5 Noun Phrase Extraction

The next step in the pipeline is to detect common mentions of phrases that could entail cloud service features. The expression of a specific feature or functionality could vary across different sources, in terms of syntax and choice of words, however, there should be semantic similarities and patterns that could be recognized in order to classify a phrase as a feature.

Noun phrase extraction is a valuable technique in natural language processing that involves identifying and extracting noun phrases (groups of words that function together as nouns) from unstructured text. Its application in this context is that noun phrases often contain the core ideas or entities in a text. These can include specific features, components, tools, or technologies. Looking at some of the terminology and the central nodes found in the NIST cloud taxonomy also backs this claim. Some examples are:

- Service Orchestration
- Information Privacy
- Security Audit
- Hybrid Cloud
- Content Management

All these names are considered noun phrases, and the derivation of new features requires the detection of commonly used phrases on different vendors' websites. This is done through POS tagging and chunking, which are already defined in Appendix B, as they also appeared in the Implementation Features analysis (3.8.1). In their majority, shared NLP libraries such as SpaCy and NLTK provide out-of-the-box implementations of these techniques. Through carrying out the design cycle of the Design Science methodology (2.3), these libraries were tested and found to be inadequate (despite their computational efficiency) for this stage of the pipeline for several reasons. The first problem found was the inconsistency with which POS tags are assigned to tokens, which tampered with the process of chunking noun phrases. Examples of phrases that failed to get grouped because of their tags are:

- Load Balancing
- Auto Scaling
- Machine Learning
- Web Hosting

The problem with such terms is that they often include verbs, like the words "balancing" or "scaling", which causes the noun chunker to split them. The best-performing tool found for this purpose was the Sequence Tagger from the *flair*[13] library. Flair's tagger utilizes a BiLSTM (Bidirectional Long Short-Term Memory) network combined with CRF (Conditional Random Fields). BiLSTM handles the input sequence in both forward and backward directions, ensuring that each word's context is used for

[13] https://huggingface.co/flair/pos-english

prediction, while the CRF layer is used on top of the LSTM outputs to predict the tags of the entire sequence. This model achieved state-of-the-art performance on tasks like POS tagging and NER (Akbik et al., 2019). After the input sequences are tagged, the chunking process should group the tagged tokens

together to best capture the syntactic nuances of the specific feature names found in the cloud computing domain. For example, the cloud taxonomy that was introduced in includes the three most common cloud service models: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). If a consumer is interested in a specific service model, it is expected that the knowledge graph will also be able to categorize and display services under these service model categories or even identify newly emerged categories. However, available noun chunkers will typically split such phrases, as well as the following examples:

- denial of service (DDoS) attacks
- multiple platforms
- structured query language (SQL)
- Pay As You Go
- backup as a service

Such examples of phrases include sequences of not exclusively noun tags, but also adjectives, prepositions, and more. Therefore, a custom Chunk Grammar (definition of patterns of part-of-speech tags that determines how words should be grouped) was iteratively created and refined to capture these nuances and effectively group the tokens into phrases commonly used in this domain. Although its application could also be transferable to other software engineering domains. The phrases found for each vendor were once again de-duplicated to form the dataset for the next pipeline stage, which is clustering similar phrases together.

5.6 Phrase Embeddings

To assess the similarity of phrases, they must first be transformed into a numerical form that is eligible for computation. Embeddings are numerical vectors representing textual data in a high-dimensional space, allowing for the quantitative assessment of the similarity between texts (Mikolov, Chen, et al., 2013b). Plenty of word embedding model categories are available that are suitable for this task. One of the emergent architectures that revolutionized the field of NLP and produces very information-rich representations is the transformer architecture, which utilizes the attention mechanism to consider the entire context of a sentence or word sequence, computing the relevance of each word to the entirety of the sequence and hence providing a better understanding of language in general.

One model based on this architecture is Google's BERT (Bidirectional Encoder Representation from Transformers). Even though newer models based on the same principle have been introduced since the release of BERT, it still remains a widely used model due to its robustness and versatility in handling context-rich embeddings (Devlin et al., 2019). There are a plethora of variations of BERT models available to choose from, each of them specializing in the specific task they have been trained for. The model used in this project is **"paraphrase-TinyBERT-L6-v2"**, a smaller and more efficient version that, as the name suggests, was trained to recognise the semantic equivalence of paraphrased expressions of the same sentence. This model is available from the *sentence_ transformers* library. The intuition behind this selection is that even when similar features are described differently, they will still be grouped if they refer to the same thing.

5.7 Clustering

In order to automatically generate an ontology from text corpora, terms are first extracted as lexical representations of concepts, which are then organized into a hierarchical structure using clustering techniques (Meijer et al., 2014a). After extracting, refining, and vectorizing noun phrases, the goal is to identify which phrases are semantically similar enough to represent the same concept or, in this case, cloud service feature. Clustering is an essential and widely used technique in the field of NLP, also evident in the implementation features analysis (3.8.1). Clustering enables the grouping of data points into distinct groups, or clusters, based on their similarity. Especially in projects of this nature, where

the goal is to organize and classify unstructured text data without predefined labels, this technique provides a method to extract meaningful patterns and structures from the data, which can then inform the labeling process. Several popular clustering techniques have been widely utilized in various domains, such as k-means, hierarchical clustering, DBSCAN, and Gaussian mixture models. The most common one is K-means, which requires the manual curation of the number k, of clusters to be formed by the algorithm. This parameter also affects the final size of the clusters to achieve the required number of clusters. Since there is no way to predict the number of categories and the granularity of the features to detect in the particular case of cloud computing, an alternative algorithm is needed.

5.7.1 Hierarchical Clustering

Hierarchical clustering is a technique that seeks to construct a hierarchy of clusters, with the flexibility of not requiring to define the number of clusters beforehand. There are two types of hierarchical clustering: Agglomerative (bottom-up) and Divisive (top-down) (Xu & jie Tian, 2015).
Agglomerative Clustering: Starts with each element as a separate cluster and merges them into successively larger clusters.
Divisive Clustering: Starts with the entire dataset as a single cluster and splits it into progressively smaller clusters.
Generally, the limitation of such algorithms is the handling of noisy datasets with varying densities and outliers (Xu & jie Tian, 2015). Density refers to how closely the data points are situated in the data space. In this context, this may be correlated to how the traditional cloud service features may appear in tightly packed clusters in the vector space, with many data points close together. In contrast, new or innovative features or features that are tailored for more specific use cases may appear in more spread out clusters with fewer data points. One example of the former could be cloud security-related features, and one example of the latter could be Artificial Intelligence functionalities, a recently rapidly evolving field in any technology-related sector, prone to develop many new and diverse features.

5.7.2 HDBSCAN

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is a hierarchical clustering algorithm that mostly aligns with agglomerative clustering algorithms in terms of their bottom-up approach. A density-based clustering algorithm extends DBSCAN (Density-Based Spatial Clustering of Applications with Noise) by incorporating varying densities in the resultant clusters. As opposed to agglomerative clustering, which starts with each point as its cluster and iteratively merges clusters, primarily based on distance measures, HDBSCAN focuses on areas in the data space with different densities, therefore allowing for more variety of clusters in that aspect (McInnes et al., 2017). Another benefit of this technique is the management of noise and outliers, likely to appear in the noun phrase dataset in large amounts, as nouns are core elements of any sentence. Finally, HDBSCAN handled large datasets well, showcasing its scalability and computational efficiency. Consequently, this was the clustering algorithm of choice.

5.7.3 Principal Component Analysis

The number of data points involved in the clustering process was 189016, while the embedding vector for each data point had a size of 768. This results in a very high dimensionality of the input data to the clustering process, which inevitably causes its computation time to increase significantly. Besides the efficiency concerns, algorithms like HDBSCAN can operate more effectively in lower-dimensional spaces because noise and excess variation in the data points can deceive the grouping process. PCA (Principal Component Analysis) is a statistical dimensionality reduction technique that compresses the number of dimensions of data into a smaller number of the most informative components, called principal components, that explain the most variance in the data (Abdi & Williams, 2010). Also appearing in the component analysis (3.6.1), PCA is a popular and useful tool, not only in NLP, but in other areas of AI research where dimensionality reduction is necessary.

The number of dimensions to keep after the dimensionality reduction can be determined by fitting the PCA model to the embeddings, which involves the calculation of eigenvectors for the covariance matrix of the data that represent the directions of maximum variance (i.e. principal components) (Minka, 2000). The PCA model used is available from the *sklearn* library tailored for machine learning. The results of the above process are shown in figure 5.5.

Variance Explained	PCA Components
80%	116
85%	145
90%	191
95%	288

(a) Number of PCA Components to Explain Different Percentages of Variance

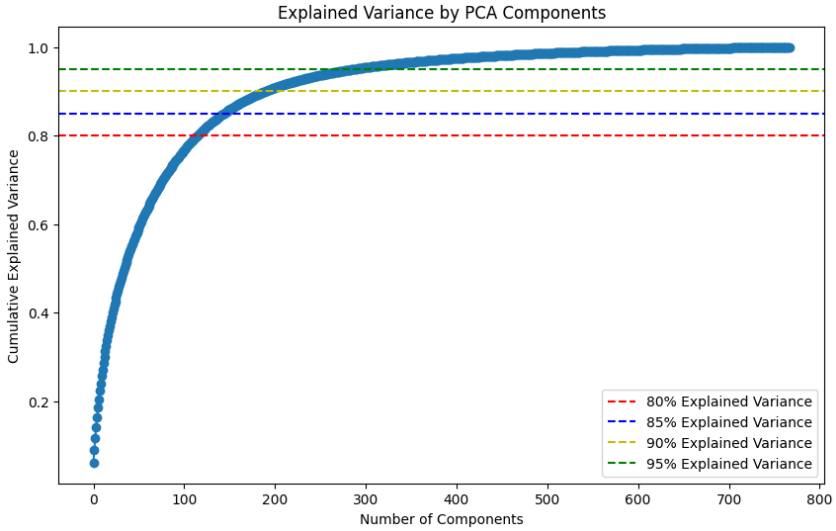

Figure 5.5: PCA Variance Explained Plot

5.7.4 Parameter Selection

When using HDBSCAN for clustering, there are two critical parameters of the algorithm to set, which greatly affect the clustering results:
min_cluster_size: The minimum number of data points required to form a cluster. Groupings smaller than this size will be considered as noise.
min_samples: The number of samples in a neighborhood for a point to be considered as a core point. This includes the point itself. This parameter influences the density requirement for a region to be considered as a cluster.
These parameters primarily affect the granularity and stability of the clusters formed. Setting the values too small could result in an excess number of small and potentially irrelevant clusters. In contrast, higher values could decrease the granularity more than intended and simultaneously combine multiple distinct cloud service features in the same clusters. The goal is to identify different groups of data points where each group represents a specific type of service feature. After iterative refinement of the parameter values and inspection of the clusters formed, this dataset's selected values were min_cluster_size = 10, min_samples = 5.

5.7.5 Clustering Results

Due to the large number of data points and the expected granularity of the results to capture the high variety and specificity of features, a significant number of clusters and a large amount of noise are expected. The number of clusters under the abovementioned parameters is **3203**. The average size of each cluster is **19**, while **128173** out of the total **189016** data points were regarded as noise. The distribution of the cluster sizes is shown in figure 5.6. We can consider each cluster as a data point in the later stage of the pipeline and the classification of each cluster as a relevant cloud service feature or category, eligible to be included in the knowledge graph. Considering that many clusters will not be selected. In contrast, others can potentially be merged if they essentially refer to the same thing, the size of the dataset appears to be sensible at first sight. It must also be noted that the clustering results are not expected to provide definite and distinct groups of features, as the purpose of this process is

to observe patterns and common mentions of domain-specific terms in the text collected from different vendor's websites, facilitating the feature identification process. At this stage, no context is used for each phrase involved in the clustering process, as the context is not always representative of what each phrase means, especially when specific technical terms are merely mentioned and not explained, even in the particular sentence they are found in.

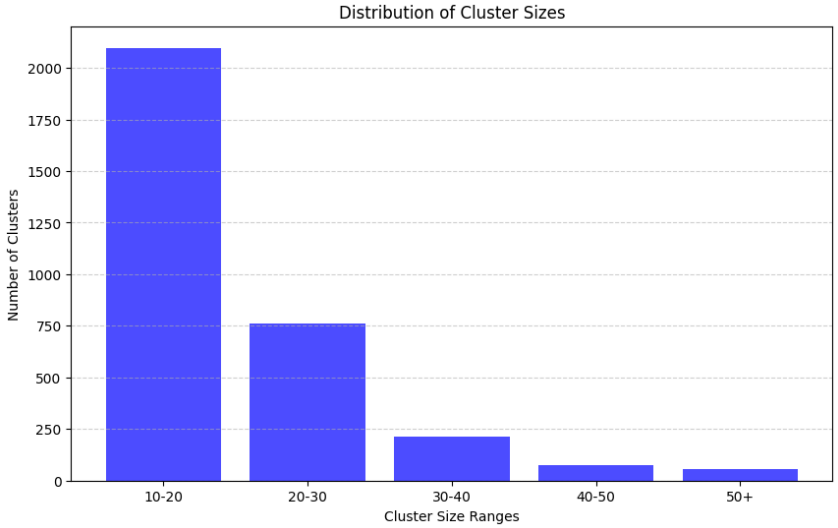

Figure 5.6: Cluster Size Distribution

In order to justify the parameter selection with quantitative metrics, besides the qualitative manual inspection of the consequent cluster, relevant numerical metrics may be employed. The most conventional metric for the evaluation of clustering algorithms is the Silhouette Coefficient. It is calculated using the mean intra-cluster distance and the mean nearest-cluster distance for each sample, to provide a measure of how similar a data point is to the data points of the same cluster (cohesion), compared to the data points of other clusters (separation). The value ranges from -1 to +1, where an average silhouette score of over 0.7 is considered excellent, and values around 0.5 are satisfactory. In contrast, lower values show weak cohesion and separation or are even poor for values lower than 0. With increasing dimensionality of the data, high values are less achievable because of the curse of dimensionality, as the distances to points of the same cluster become comparable to the distances to points in other clusters.

Another metric that also evaluates clustering quality, but from a slightly different angle, is the Davies-Bouldin Index (DBI), defined as the average of the maximum ratios of the sum of intra-cluster (within clusters) distances to the inter-cluster (between clusters) distance. The values for this index do not have a specified range, but lower values indicate better cluster quality. Table 3 shows the calculated values for both the Silhouette coefficient and DBI for three sets of values for the **min_cluster_size** and **min_sample** parameters. The values are given in 3 s.f. The chosen values are (10, 5) and aim to strike a healthy balance of feature granularity with cluster cohesion and stability.

Parameter Set	Davies-Bouldin Index	Silhouette Score
(10, 5)	0.978	0.505
(20, 15)	0.939	0.483
(30, 20)	1.011	0.441

Table 3: Evaluation Metrics for Different Clustering Parameters (min_cluster_size, min_sample)

5.8 Cluster Aggregation

Each phrase cluster should represent a single potential cloud service feature. At the same time, it is possible that clusters are not entirely uniform and free from noise or data points that may have been

wrongly included in specific groups due to subtleties and slight variations in technical terms. Therefore, a method is required to aggregate each cluster into a single name, most representative of the phrases constituting the cluster. In addition to representation, the derived names of clusters are essential for a more straightforward classification of features and indexing, presentation, and retrieval purposes from the resultant knowledge graph.

5.8.1 N-grams

Generally, an n-gram is defined as a collection of n successive terms in a piece of text. Clustered phrases may describe similar concepts expressed in different ways. For instance, the phrases "data backup services" and "services for data backup" obviously refer to the same feature: "data backup", which also happens to be the common n-gram (where n = 2) between these phrases. By analyzing the frequency of n-grams within each cluster, for different values of n, common patterns can be identified that capture the core of the cluster semantically.
The logic for selecting the most representative n-gram involved the following steps:

- Pre-process the phrases set to normalize the nuances, such as capitalization and punctuation. For example, ensuring equivalence between "open-source software" and "Open Source Software".
- Compute the frequency of all n-grams in the set of phrases of each cluster for values of n of 1 up to 7.
- Calculate the proportion of vendors that mention each n-gram out of the set of vendors associated with the cluster, applying a threshold to ensure that at least half the vendors use that n-gram in their vocabulary for a specific feature.
- Starting from the shortest and most common n-gram found, move up levels in terms of the n value to extend the result to a longer n-gram without losing relevancy, i.e., check for eligibility of the newly introduced words based on the vendor coverage metric introduced in the previous step. If no longer n-gram is eligible, the search ends with the current result.

This procedure considers the statistical significance of phrase occurrences and the required representation across the vendor inputs that form each cluster.

5.8.2 Examples of Cluster Aggregation

The following examples of clusters and their corresponding derived names showcase the effectiveness of the method described above.

Cluster: ["fault tolerance of the system", "inherent fault tolerance", "fault tolerance", "built-in fault tolerance", "Fault tolerance", "Fault Tolerance", "fault tolerance", "stronger fault tolerance", "fault tolerance", "FAULT TOLERANT", "fault tolerance", "best fault tolerance", "high fault tolerance", "fault tolerance against downtime", "Fault Tolerance", "high fault tolerance", "fault tolerance and durability", "level of fault tolerance"]
Derived Name: "fault tolerance"
Cluster: ["SSO to AWS applications and resources", "SSIS", "SSI", "SSIS", "SSO", "SSO", "SSO", "SSO", "SSO organization connect for Exoscale", "SSO applications"]
Derived Name: "SSO"

In the second example, the term "SSO" refers to single sign-on, an authentication scheme. The red-highlighted phrases are examples of noise within the cluster.

5.9 Dataset Creation

After the clustering process, the core of the dataset used to classify cloud service features and create the knowledge graph is formed. Besides the feature classification task, an additional classification task, called Match Quality Classification, is carried out to match the identified features to DBPedia resources, enabling the enhancement of the knowledge graph in a multitude of ways:

- Provide concrete explanations for specific features through the DBPedia abstracts in any matched resources.

- Coreference Resolution: merge clusters that are matched onto the same resources (i.e. potentially refer to the same concept/ feature), into single feature nodes in the graph.
- Implicit knowledge extension: by incorporating the DBPedia resources in the knowledge graph, additional information can be directly accessed for each feature regarding its categorization and relationship to other concepts through the RDF framework of the semantic web.
- Ontology Creation: the creation of the final hierarchical structure of the knowledge graph can be informed by taking advantage of already existing relationships between the graph nodes through the DBPedia Ontology namespace.

The match quality classification succeeds the feature classification in the conceptual pipeline. This is because no information from DBPedia is intended to be used in the initial identification of features to prevent potential mismatches from altering the performance in the first task. Only the data directly extracted from the vendors' web sources is used.

5.9.1 Feature Classification Dataset

This is the primary classification task involved in the pipeline. As previously mentioned, each phrase cluster can be considered a potential cloud service feature and, therefore, a member of the relevant dataset for this task. At this point, it is reasonable to loosely set the standards for what a feature is. The goal is to identify terms that can be considered useful and informative in the context of vendor/ service selection, as well as their categorization in the hierarchical structure of the knowledge graph. For example, the following content items are found in the content list of a particular cluster:

'Fully managed service that helps secure remote access to your virtual machines.'
'Gain unified, remote access with single sign-on across on-premises, infrastructure as a service and software-as-a-service-applications.'
'VPN options are available for remote access.'
'As remote access needs grow, organizations are increasingly shifting away from traditional VPN implementations and toward more secure and performant remote access solutions.'

The derived name of this specific cluster is "remote access", which is a somewhat general term that can be found in various contexts in the field of cloud computing since the "cloud" is generally considered a network of remote servers that are accessed over the internet. Therefore, the term "remote access" does not satisfy the criteria above to be included in the knowledge graph. On the other hand, the last two of the listed sentences also mention the term "VPN", short for Virtual Private Network, a common offering of most cloud vendors which entails a more specific aspect of remote access services and could potentially be eligible for inclusion.

5.9.2 Integrating Context into Classification

Looking back at the last content item, it can be deduced from the provided context that VPNs are a remote access solution. This is just an example of how the context in which a term is used can drastically influence the interpretation of its meaning and relevance. It also mitigates the effects of linguistic nuances such as polysemy, i.e., when a word can have multiple meanings depending on its context. For predictive models, this provides a better representation of the data, resulting in more accurate predictions. Given the variability in terms of specificity and relevance across phrases found in cluster content items, it is essential to incorporate content items directly into the input of the chosen classification algorithm.

A challenge presented in doing this is handling the volume of context information efficiently. With an average cluster size of 19, it would be computationally intensive to include all the content items in which the phrases of each cluster are found. It could also dilute the focus from the term being classified, especially when, in some cases, they are surrounded by multiple other terms, and the context does not necessarily explain the term of interest. Thirdly, as previously seen, some of the members of each cluster are noisy and do not allude to the same thing as the rest of the cluster. Consequently, a strategy must be employed to select a subset of the content items. The following criteria were considered to choose two

content items for each cluster and form the input data point to the feature classifier by concatenating the selected context to the derived name of each cluster.

Criteria for Selection:

1. **Frequency:** Preference should be given to the context of phrases that appear frequently across the cluster, indicating their prominence and acceptance. This is already enforced through the cluster aggregation method described in 5.8 by choosing sentences that contain the derived name of the cluster since it embodies the most common n-gram found in the cluster. This also eliminates the possibility that a context of some noise phrase will be selected.
2. **Simplicity and Focus:** Content items containing fewer noun phrases besides the clustered phrase are prioritized to select sentences that are more straightforward and focused on the target phrase. This approach is based on the assumption that such sentences would provide a more direct context for the phrase without as many additional and potentially distracting terms.
3. **Computational Intensity:** When processing large datasets or implementing models where complexity significantly impacts performance, shorter sentences are expected to reduce the computational load and because of the simplicity of the inputs.

It must be noted that this approach also entails the risk of oversimplification and loss of contextual information. There is room for modifications and improvement of the context selection strategy. However, in this project's scope, where the focus is on examining the feasibility of the recommended system, even with limited computational resources, more gravity was given to keeping the input complexity at a lower level. Furthermore, the quality of the available context from the web sources used to collect the data is not a controlled variable, so the system's functionality under this circumstance further demonstrates the robustness of its proposed architecture.

5.9.3 Match Quality Classification Dataset

The curation of the dataset for this second classification task involves the selection of DBPedia resource URIs as potential matches to each phrase cluster, where each cluster may potentially match with none, one, or more than one DBPedia entry. The list of the candidate URIs for each cluster is formed using a manifold method.

Initially, the URIs stored from the process described in section 5.4.2 are considered, referred to as the original DBPedia vocabulary. The same embedding model used to encode the phrases before the clustering process is also used to encode the URIs' labels (or names). The most similar data point involved in the clustering is found by computing the cosine similarities between the two sets of embeddings. The purpose is to identify which cluster, if any, best suits each of the URIs without re-clustering. The URIs that pair with clustered phrases instead of phrases regarded as noise indicate which DBPedia entries align better with the patterns found in the data. Thus, they are used to expand the vocabulary by collecting information on all the DBPedia resources linked from their Wikipedia pages through the "dbo:wikiPageWikiLink" predicate.

The expanded list of DBPedia resources is again checked for similarity with the clusters; this time, however, against all the phrases constituting them. If the average cosine similarity with the phrases of each cluster exceeds a specified threshold, the URI is added to the list of potential matches for that specific cluster.

The final way that candidate DBPedia resources are collected for each cluster is through the DBPedia API[14], where the derived name of each cluster can be used as a keyword to search for relevant entries. The names of the returned resources are once again encoded, compared to the embeddings of the cluster phrases. The resource with the highest average similarity is added to the dataset, given its similarity exceeds the same threshold applied before.

5.9.4 DBPedia Context Integration

The second classification task is an example of entity resolution, where the goal is to identify whether different textual representations describe the same feature, concept, or entity. To achieve this, we need

[14] `http://lookup.dbpedia.org/api/search`

to utilize and compare the context from both the web and DBPedia sources to assess their similarity. For each cluster, the context used in feature classification, which includes the most representative content items, is leveraged again for the match quality classification, along with the derived name of the cluster from DBPedia, not only the name of but also the first sentence of the abstract of each entry is used, which typically provides a compact definition or description, making it a computationally efficient choice to grasp the essence of the entry. This will allow the classifier of choice to make more informed and accurate decisions.

6 Classification

One of the objectives of this project was to showcase the feasibility of automatically identifying software features using unsupervised techniques, which require no manual labor or data labeling. While supervised learning heavily relies on labeled data to train models, unsupervised learning operates without such labels, making it a challenging approach.

Clustering algorithms are considered essential tools for unsupervised data labeling and enable the grouping of data points into distinct categories based on their similarity, as also implemented in the system architecture described so far. Beyond this point, however, the classification of the data points in each group to more practical and structured representations of real-world concepts, as in a knowledge graph, is usually a labor-intensive task.

Text classification is a task that has been extensively researched in recent years, as it has been tackled with unprecedented success due to the rise of Large Language Models (LLMs) like BERT, GPT (Generative Pre-trained Transformer) and many other variants (Minaee et al., 2021). This is also evident by the prevalence of such models in the analysis given in 3.6.1. These models are particularly proficient at comprehending human language since they are extensively pre-trained on numerous corpora of text, enhancing their contextual understanding of words and how they are used in various domains. Usually, pre-trained models are used in transfer learning scenarios, where they are fine-tuned on task-specific datasets and exploit their general knowledge about language to optimize for those tasks, such as text classification.

Without labeled data, classification models are trained under different learning setups. An example is active learning, which involves selecting the most informative instances from the unlabeled dataset to be labeled by expert annotators in an iterative feedback loop where the model trains on an expanding labeled dataset, requests from the human annotator to label the data for which it made the least confident predictions, and retrains with more labeled data each in each iteration (Schröder & Niekler, 2020).

Another learning method, which does not require any manual intervention, is few-shot learning, which involves training models with a small amount of labeled data, with the prerequisite that the model can generalize sufficiently well with just a few examples per category or label. A more extreme case of this method is zero-shot learning, where models are trained on various tasks and expected to generalize well to new tasks without using any labeled data. While research on zero-shot and few-shot learning is progressing steadily to unburden developers of AI systems from the inefficient requirement of manual data labeling, these methods typically do not yet achieve the same level of performance as supervised learning, given adequate labeled data is available (Y. Wang et al., 2020). However, with the recent scale-up in the size of language models, their ability to perform classification with only a few examples emerged (Kaplan et al., 2020), with practical implications in scenarios where labeled data is scarce or expensive.

6.1 Overview of Experiments

The experiments that will be carried out for this project will test, compare, and present the capabilities of language models for the classification tasks and prepared datasets described in the previous chapter. Two scenarios will be used to assess the practicality of keeping manual intervention as low as possible. In the first, different variations of the BERT model will be trained in a supervised fashion to provide a preliminary benchmark for the classification performance using supervised learning. Note that no expert annotation is employed for what the expected predictions and consequent list of cloud service features should be, along with their matched DBPedia entries, due to the limited scope and resources of the project. The second approach is to use a generative LLM, specifically Google's Gemini, along with prompt engineering, to implement a few-shot direct prediction of the labels for both classification tasks, allowing the comparison of this method with the set benchmark.

6.2 NIST Labeled Data

For the few-shot classification method mentioned above, a small amount of labeled data is required to guide the process, as a few examples must be provided to the generative language model through the composed prompts to allow it to understand the task at hand better, the expected output, as well as the targeted nature of features that need to be classified, in regards the feature classification task. The NIST

document introduced in 5.4.1 can be considered a concrete, reliable source for such examples since it contains terminology that can be mapped to a portion of the clusters, hence labeling them as relevant. At the same time, the concepts described in the document are also organized in the provided cloud taxonomy, which enables us to kickstart the ontology creation following the identification of the relevant features by manually adding in the features associated with the labeled clusters and their proposed parent-child relationships. A total of 201 clusters were ultimately labeled and excluded from the classification process, as well as their matched DBPedia resources.

6.3 Evaluation Metrics

The standard evaluation metrics for classification tasks, such as Precision, Recall, Accuracy, and F1 score, will be used throughout the experiments. To compensate for the class imbalance (i.e., the disparity in the number of true labels and false labels in a given set of data samples) during both the training and testing phases, the weighted average of the precision, recall, and F1 scores are calculated so that the contribution of each of the labels to the overall metrics is proportional to their number of instances in the selected dataset.

Precision is defined as the proportion of predicted positive instances that are indeed positive:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{1}$$

Recall is slightly different, calculating the proportion of true positives with respect to the total of true positive instances:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2}$$

The F1 Score combines both the precision and recall metrics to compensate for the imbalance in the number of true positive and true negative instances:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

Finally, the Accuracy metric is the proportion of correct predictions out of the total predictions made:

$$\text{Accuracy} = \left(\frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \right) \times 100 \tag{4}$$

6.4 Experiment 1: Supervised BERT Models

A total of 5 BERT models were trained on the two classification tasks. The selected models are:

- **BERT-base-uncased and BERT-large-uncased:** BERT-base is a broadly used BERT model with 12 transformer layers and 110 million parameters. It offers a good starting point for text classification, striking a balance between performance and resource requirements. On the other hand, BERT-Large has 24 transformer layers and 340 million parameters, enabling it to capture more complex contextual information, although it requires significant computational resources for training and inference.
- **RoBERTa-base and RoBERTa-large:** robustly optimized versions of BERT built with modified key hyperparameters and allegedly better pretraining techniques. These models have demonstrated better performance in tasks requiring determining the relationship between two sentences, which may be useful for the match quality classification.
- **distilBERT-base-uncased:** a distilled version of BERT, which offers a good baseline comparable to the full BERT model while being smaller and faster. This model is ideal for limited resource environments.

The use of both base and large versions of models aims to display the effect of model size on performance. The base models offer a standard comparison point, while the large models test the benefits of increased complexity and number of parameters on classification accuracy.

6.4.1 Experimental Setup

The dataset was split into a train set and a validation set, with the ratio 80:20, a standard practice in machine learning to balance the availability of data to train while having sufficient data to test the performance of the models. Due to the small size of the datasets, three training epochs were found adequate to reach peak performance. More significant numbers were also tested, specifically 5 and 10, which led to decreased performance on the validation set, indicating the effect of overfitting. The batch size was set to 16, as smaller batches can provide better generalization when the dataset is small. The typical loss function for binary classification tasks was used, i.e., binary cross-entropy (BCE), which quantifies the difference between the predicted probability distribution and the true distribution of the labels and can be mathematically expressed as:

$$\text{BCE}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

where:

y_i is the true label for the i-th sample.
$\hat{y}_i$ is the predicted probability for the i-th sample.
N is the total number of samples.

6.4.2 Results

Table 4 shows the classification performance metrics for the five models, where P stands for Precision, R for Recall, and A for Accuracy. As demonstrated by the performance of the distilBERT-base model in the feature classification task, compared to larger models, bigger numbers of trainable parameters of a model do not always result in better results. The small dataset size might impose this, as larger models require more data to capture the underlying patterns in the data effectively. At the same time, they are more prone to overfit the training data if the right regularization and optimization techniques are not applied, causing a drop in performance for the validation data.

Even though the goal of this experiment is not to optimize the classification performance, as stated in 6.1, the interpretation of the results depends on the emphasis given to specific requirements of the system. For example, if the system is required to collect as much valuable information (or features) as possible, recall indicates whether most of the features are identified. This might entail including many insignificant features, however, if the precision is not very high. The F1 score may indicate a healthy balance between the two.

Table 4: Classification Metrics for Supervised Learning

	Feature Classification				Match Quality Classification			
	P	R	F1	A	P	R	F1	A
BERT-base	0.82	0.81	0.81	0.81	0.87	0.86	0.86	0.86
BERT-large	0.79	0.74	0.74	0.74	0.87	0.87	0.87	0.87
RoBERTa-base	0.74	0.74	0.74	0.74	0.75	0.76	0.73	0.76
RoBERTa-large	0.83	0.81	0.81	0.81	0.82	0.81	0.79	0.81
distilBERT-base	0.82	0.81	0.82	0.81	0.85	0.85	0.85	0.85

6.5 Experiment 2: Gemini Classifier

Google's Gemini is the generative language model used in the experiment, mostly because of its easy accessibility through the Google Cloud API. A comprehensive list of alternative generative models and their features is found in the spreadsheet called "LLMs" from the general SLR documentation spreadsheet (Constantinou, 2024). Gemini's core utilizes a transformer-based architecture, making it suitable for

tasks that require an advanced understanding of language and contextual information, such as few-shot classification. The method of prompt engineering was employed to iteratively improve Gemini's performance in the two classification tasks.

6.5.1 Prompt Engineering

Prompt engineering is designing and refining textual inputs to generative language models that direct it to generate the desired output in content and format. It can also be applied for text classification, where the input text to be classified is included in the prompt, along with a description of the task and a few examples to inform the model, using examples, about the target labels and what sort of input maps to each label (Brown et al., 2020). The examples should, therefore, be as broad and representative of the whole dataset as possible, but keeping the prompt clear and specific is also essential.

6.5.2 Example Prompts

The following prompts are the primary prompts used to test the performance of this technique. By checking the accuracy of the model's responses, two parts of the prompt were iteratively refined: the description and the examples. In the first task's prompts, the variable "term" is replaced by the derived cluster name as computed using the n-gram technique, while "contexts" are the sentences chosen to represent the web context in which the term appears. For the second task, "term2" and "contexts" take the same values, while "term1" and "abst_sent" are the DBPedia resource name and the first sentence of its abstract, respectively. The results in the following section convey the gradual improvement in the classification performance by refining the input prompts.

Feature Classification Prompt 1

```
Considering cloud service selection and the few-shot examples below,
classify Term, as seen in Context: Relevant (True/False)?

example: CRM
Context: Choose a CRM implementation partner that continues to grow
and adapt with you, helping you stand out from the rest. / This CRM
was designed for small business.
Relevant: True

example: audit logs
Context: Likewise, you can store audit logs for accessing the
Kubernetes API in an S3 bucket you define. / With Cloud Audit Logs
integration, every interaction with Secret Manager generates an audit log.
Relevant: True

example: resources
Context: Distribute traffic across resources / All resources are priced
on an hourly basis, so you'll only pay for what you need, when you need it.
Relevant: False

example: scratch
Context: Create custom solutions without starting from scratch. / No
need to start from scratch.
Relevant: False

Term: {term}
Context: {" / ".join(contexts)}
Relevant? Answer with one word
```

Match Quality Classification Prompt 1

```
Given two terms and the context they appear in, classify if they refer
to the same thing/ concept.

Examples:
  * Term 1: Scratch (programming language)
    Context 1: Scratch is a high-level block-based visual programming
    language and website aimed primarily at children as an educational
    tool for programming, with a target audience of ages 8 to 16.
    Term 2: scratch
    Context 2: Create custom solutions without starting from scratch. /
    No need to start from scratch.
    Answer: False

  * Term 1: CRM software
    Context 1: Customer relationship management (CRM) is a process in
    which a business or other organization administers its interactions
    with customers, typically using data analysis to study large amounts
    of information.
    Term 2: CRM
    Context 2: Choose a CRM implementation partner that continues to grow
    and adapt with you, helping you stand out from the rest. / This CRM
    was designed for small business.
    Answer: True

  * Term 1: Hypervisor
    Context 1: A hypervisor (also known as a virtual machine monitor,
    VMM, or virtualizer) is a type of computer software, firmware or
    hardware that creates and runs virtual machines.
    Term 2: hypervisor
    Context 2: Regardless of which hypervisor you choose, we manage and
    support your virtualized configuration, 24x7x365. / Hypervisor :
    KVM on Linux
    Answer: True

  * Term 1: Integrity
    Context 2: Integrity is the practice of being honest and showing
    a consistent and uncompromising adherence to strong moral and
    ethical principles and values.
    Term 2: integrity
    Context 2: With business data scattered across the enterprise
    and value chain, managing and maintaining its integrity can be
    difficult. / Ensuring your data and environments are properly
    segmented from other machines is extremely important for the
    integrity of your data.
    Answer: False (data integrity is not exactly the same with general
    integrity described above)

Term 1: {term1}
Context 1: {abst\_sent}
Term 2: {term2}
Context: {" / ".join(contexts)}

Answer? Provide single wordedly with True or False
```

6.5.3 Results

The following tables (Table 5 and Table 6) present the classification metrics for the few-shot prompting technique applied to the two classification tasks: Feature Classification and Match Quality Classification. For each prompt, metrics are displayed in two sets: the first set, on the left-hand side, shows the computed metrics on the validation set used for the supervised learning models, enabling a direct comparison. The second set, on the right-hand side, lists the metrics computed across the entire datasets for both tasks, providing a comprehensive assessment of the model's overall performance. The assessed prompts are fully displayed in appendix E.

Table 5: Few-Shot Prompting for Feature Classification

	Validation Set				Entire Dataset			
	P	R	F1	A	P	R	F1	A
Prompt 1	0.60	0.61	0.57	0.61	0.59	0.61	0.57	0.61
Prompt 2	0.83	0.82	0.82	0.82	0.82	0.81	0.81	0.81
Prompt 3	0.83	0.77	0.77	0.77	0.82	0.76	0.76	0.76
Prompt 4	0.81	0.70	0.69	0.70	0.81	0.70	0.69	0.70
Prompt 5	0.82	0.79	0.79	0.79	0.77	0.74	0.75	0.74
Prompt 6	0.91	0.91	0.91	0.91	0.81	0.80	0.81	0.80

Table 6: Few-Shot Prompting for Match Quality Classification

	Validation Set				Entire Dataset			
	P	R	F1	A	P	R	F1	A
Prompt 1	0.73	0.72	0.72	0.72	0.72	0.71	0.71	0.71
Prompt 2	0.82	0.68	0.69	0.68	0.82	0.68	0.69	0.68
Prompt 3	0.83	0.73	0.74	0.73	0.82	0.73	0.74	0.73

Overall, the results indicate that the performance of the few-shot prompting technique on the Feature Classification task is comparable to that of supervised learning models, such as the BERT variants. This suggests that the Gemini model's enhanced contextual understanding can be effectively leveraged in scenarios where the task description is sufficiently specific and detailed, allowing the model to exploit its pre-trained knowledge. In contrast, the Match Quality Classification task demonstrated less promising results with the few-shot technique, performing below the level of the BERT models.

A plausible interpretation of these outcomes is that the Feature Classification task inherently benefits from the contextual understanding of the Gemini model, particularly when the task is explicitly described and supplemented with clear examples. BERT models, constrained by the limited training data available, may struggle to encode such complex contextual information into their weights. Conversely, the Match Quality Classification task, which primarily involves semantic similarity determination, is more straightforward and does not extensively capitalize on the advanced contextual capabilities of the generative model, thereby favoring the more direct training approach of the BERT models.

Notable Observations

Several key factors were observed to influence the effectiveness of the few-shot prompting technique:

- **Importance of Example Quantity and Quality:** The number and quality of examples significantly affect the model's performance. For instance, providing opposing examples, such as differentiating between "Bytes" and "Storage Capacity," helps the model better understand the distinctions between related concepts.
- **Balance of Positive and Negative Examples:** A balanced representation of positive and negative examples within the prompt is crucial to avoid model bias and ensure a more robust understanding of both relevant and irrelevant classifications.

- **Complexity and Specificity of Instructions:** The complexity of the prompt instructions must strike a balance between being sufficiently explanatory to guide the model effectively and avoiding unnecessary details that may introduce confusion. Instructions should be clear, concise, and directly relevant to the classification task at hand.

The importance of carefully designed prompts in leveraging the potential of few-shot learning is underlined by the above observations, especially in tasks requiring intricate understanding and interpretation. Prompt design could be optimized further to enhance the classification performance of generative models in future work.

6.5.4 Summary of Results

The experimental results in this section showcase the comparative effectiveness of two approaches: supervised learning and few-shot classification. In the supervised learning experiment, BERT models were trained on a limited amount of labeled data demonstrated good classification performance for the feature classification task, with a bottleneck of F1-score at around 0.82. This suggests that with adequate labeled data, supervised methods can effectively capture the nuances of cloud service features, leading to reliable categorization. However, this approach's dependence on labeled data limits its scalability in contexts where labeling resources are scarce.
The few-shot classification experiment offered promising results, showing that models could adapt to new examples with minimal labeled data. On the same validation set, this method achieved incrementally better results as the prompts are refined, reaching an even better classification performance (F1-score of 0.91), underscoring the flexibility and potential of this method for handling evolving feature sets in cloud services. These findings point to the utility of few-shot learning for domains with limited labeled data or rapidly changing categories, where supervised learning may struggle to maintain relevance.
It is important to note that both of these methods have potential for even better performance. Both BERT and Gemini are LLMs that have not been pretrained on domain-specific text. An intermediate pretraining step on a corpus of cloud-related documents (e.g., cloud service descriptions) could increase their classification performance (Gururangan et al., 2020). This involves training on unlabeled cloud service texts using masked language modeling (MLM), which requires no labeled data. Domain-adaptive pretraining adjusts the model's language understanding closer to the vocabulary and context specific to cloud services, or any other software domain for which the artifact is used for. However, there is additional uncertainty in terms of where these text resources can be found for any domain, as well as their reliability and sufficiency, which can impact the generalizability of this method.

7 Knowledge Graph Construction

The final stage of the pipeline outlined in section 5.2 entails the creation of the knowledge graph, defined as a structured representation of knowledge in the form of nodes and edges, where nodes represent distinct entities/ concepts and the edges represent their relationships. In our case, the knowledge graph will be populated by the information obtained from the web sources regarding the identified cloud service features (nodes) from the preceding classification phase. The targeted structure of the knowledge graph is hierarchical, where feature categories are organized into broader parent categories and narrower subcategories, reflecting their relationships and dependencies. This hierarchical arrangement aims to provide a clear and intuitive representation of cloud service features, allowing for easy navigation and retrieval of relevant information. It is important to note here that the inclusion of a vendor in any of the derived feature nodes does not itself claim the supportability of that specific feature by the vendor, but rather the mere mention of that feature in one of their web pages, which contributed to the formation of the cluster that was classified as a feature.

7.1 Ontology Creation

To construct high-quality ontologies, a substantial amount of knowledge is necessary (T. C. Du et al., 2009). The collected knowledge is the source from which the term extraction occurs to determine the building blocks of the ontology, referred to as nodes. Even when the needed knowledge is accessible, correctly organizing many concepts is still labor-intensive. Thus, finding methods to automate the ontology creation process is highly desirable (Tam, 1993). As mentioned in 5.4.1, the taxonomy included in the NIST Cloud Service Reference Architecture document was used as a blueprint to manually create the foundation of the ontology upon which the knowledge graph is built. A similar approach was taken in (Alfazi et al., 2015), where NIST Cloud Computing standards were interpreted to derive concepts, with “is a” as the primary relationship type and starting with "cloud computing" as the root node. Since the central root concepts are already present, an iterative approach may be taken to find appropriate parent nodes for each newly identified concept or feature by introducing novel parent-child relationships.

Several intricate methods have been researched to infer such relationships, for example, Formal Concept Analysis (Cimiano et al., 2005), which groups objects based on their attributes. An alternative is the subsumption method, which constructs broader–narrower relations based on the co-occurrence of concepts (Meijer et al., 2014b). Classification methods have been suggested for adding concepts to existing hierarchies, such as the tree-descending algorithm, where a term is inserted by descending from the root to the leaf and is added as a child of the leaf node on the path with the highest cumulative similarity to the new concept (Pekar & Staab, 2002).

Prior to determining the concept relationships, one of the key tasks involved in ontology creation is Coreference Resolution (Kertkeidkachorn & Ichise, 2017), which ensures all mentions referring to the same entity are identified and grouped together. This is also essential in this scenario where clusters represent the same concept and, therefore, need to be aggregated into a single node, for example:

- *Single Sign On* with *SSO*
- *Identity and Access Management* with *IAM*

Some of the information made available from the previous stages in the pipeline was used for this specific purpose, such as when the clusters associated with the same feature match the same DBPedia resource or another resource name that redirects to it, thus handling linguistic nuances such as abbreviations. Moreover, some clusters were similar despite being separated in the clustering process. Using different density values, the HDBSCAN algorithm enabled further merging of clusters to be associated with the same node in the knowledge graph. This process resulted in the final set of nodes in the graph. As previously mentioned, the next step is to organize these nodes in a hierarchical structure by expanding the NIST-based taxonomy. The contextual information from both the web text, in which the terms are found, and the textual descriptions of concepts available from DBPedia in the form of abstracts were exploited to complete this task. Specifically, the average cosine similarity of the text embeddings corresponding to the clusters associated with each node was computed to iteratively add child nodes to the existing nodes in the taxonomy, as explained in the following section.

7.2 Hierarchy Creation

The following steps were taken to determine the hierarchy of nodes in the knowledge graph and summarize the algorithm in Figure 1:

- **Initialization:**
  - N_{child}: Set of current child nodes already having a parent node in the hierarchy.
  - N_{orphan}: Set of orphan nodes that do not have a parent node in the hierarchy, excluding the central node of Cloud Computing.
- **Similarity Calculation:**
  - For each child node n_c in N_{child}, the algorithm retrieves its associated clusters C_{n_c} and corresponding embeddings E_{n_c}.
  - It then computes cosine similarity scores between the embeddings of each child node E_{n_c} and those of each orphan node E_{n_o} to determine their closeness.
- **Parent-Child Matching:**
  - The algorithm identifies the orphan node n_o^* with the highest similarity score s_{n_c,n_o} for each child node n_c and stores this pair in S.
  - The top matches T are selected based on the highest similarity scores, and potential parents are determined based on these matches.
- **Hierarchy Update:**
  - The hierarchy H is updated iteratively by adding new parent-child pairs until all orphan nodes are assigned a parent.
  - The sets N_{child} and N_{orphan} are recalculated after each iteration to reflect the updated hierarchy.

Algorithm 1 Iterative Assignm ent of Parent Nodes to Orphan Nodes in a Hierarchical Knowledge Graph

Require: Cluster data for each feature C_f, current hierarchical relationships H, and feature embeddings E_f
Ensure: Updated hierarchical relationships H

Initialize current child nodes $N_{child} \leftarrow \{n \mid n \in N \land n \in H_{child}\}$
Initialize orphan nodes $N_{orphan} \leftarrow \{n \mid n \in N \land n \notin H_{child}\}$
while $N_{orphan} \neq \emptyset$ **do**
    Initialize similarity scores $S \leftarrow \emptyset$
    for each node $n_c \in N_{child}$ **do**
        Retrieve clusters C_{n_c} for node n_c
        Retrieve embeddings E_{n_c} for clusters C_{n_c}
        if $E_{n_c} = \emptyset$ **then**
            Continue to next node n_c
        end if
        for each orphan node $n_o \in N_{orphan}$ **do**
            Retrieve clusters C_{n_o} for orphan node n_o
            Retrieve embeddings E_{n_o} for clusters C_{n_o}
            if $E_{n_o} \neq \emptyset$ **then**
                Compute cosine similarities between E_{n_c} and E_{n_o}
                Calculate average similarity score s_{n_c,n_o}
            else
                Set similarity score $s_{n_c,n_o} = 0$
            end if
        end for
        Identify orphan node n_o^* with highest similarity score s_{n_c,n_o}
        Store pair and score $(n_c, n_o^*, s_{n_c,n_o^*})$ in S
    end for
    Select top matches $T \subseteq S$ with highest scores
    for each match $(n_p, n_c, s) \in T$ **do**
        Determine best parent node n_p for child node n_c based on hierarchy depth
        Update potential parent set P with chosen pairs (n_p, n_c)
    end for
    for each orphan node $n_c \in P$ **do**
        Update hierarchical structure H with new relationship (n_p, n_c)
    end for
    Recalculate N_{child} and N_{orphan} based on updated hierarchy H
end while

7.3 Evaluation of the Knowledge Graph

A standard approach for assessing automatically generated ontologies involves a golden standard evaluation, where the constructed ontology is compared against a predefined benchmark ontology. However, in cases where no benchmark ontology exists for a particular domain, an alternative method is to have several domain experts' manual evaluation of the constructed ontology (Meijer et al., 2014b).

In this project, the chosen method for evaluating the knowledge graph's structure is by comparing the resultant set of nodes and their associated vendors with the findings of a previous study by (Bieger, 2023). In this study, the authors have collected data on the coverage of different cloud vendors regarding a predefined set of features. These include common cloud service features such as Disaster Recovery and Virtual Machines, various types of SLAs (Service-level Agreements) like Cost Management and Optimisation, Identity and Access Management (IAM), specific data center locations, Operating Systems, Programming Languages support, and more.

By manually matching these features to the final node list in the knowledge graph, we assess whether the graph can accurately retrieve information about each feature. The main purpose of the proposed system is to retrieve information about these features while identifying emerging ones. Additionally, as an extension to the proposed pipeline, determining whether a vendor actually offers a feature mentioned on their website could be explored further through the semantic classification of the content items containing the feature of interest.

The complete list of features used for comparison is categorized in Appendix F, with those identified by the few-shot classification technique marked with a checkmark. The results (summarized in Table 7) show that most cloud services, SLAs, database support, and data center locations are correctly retrieved. In contrast, to their full extent, the system exhibits some weaknesses in identifying more technical features, such as programming languages, APIs, and operating system support.

Category	**Percentage Covered**
Cloud Services	77.7%
SLAs	68.4%
Bring your own License (BYOL)	26.3%
Supported APIs	30.8%
Programming Language Support	40.7%
Databases (out of box)	90.9%
Compliance	36.0%
Service costs (Forecasting)	66.6%
Container orchestration	42.9%
Operating Systems	32.3%
Data Center Locations	51.1%

Table 7: Feature Categories and Percentage Coverage

In the following analysis, the features successfully identified were used to make naive predictions about their support by the vendors included in both studies. Specifically, a feature mentioned on a vendor's webpage is assumed to indicate that the feature is supported. These predictions are compared to the true labels found in the previous study, resulting in the metrics presented in table 8. Additionally, the number of unique URLs crawled for each vendor is reported (found in table 2), to examine the impact of the variety and volume of sources on classification performance. There appears to be a positive correlation between the number of URLs and classification performance, suggesting that the range of discoverable features is significantly influenced by the quantity and quality of textual data collected from the web, highlighting the critical role of data collection.

Table 8: Information Retrieval Metrics

	P	R	F1	A	Unique URLs
IBM Cloud	0.93	0.17	0.29	0.44	15
Alibaba Cloud	0.87	0.49	0.63	0.56	57
Leaseweb	0.43	0.41	0.42	0.65	73
DigitalOcean	0.69	0.62	0.65	0.68	78
Oracle Cloud	0.80	0.61	0.70	0.60	128
Rackspace	0.67	0.59	0.63	0.65	132
OVH	0.62	0.71	0.66	0.61	182
Amazon (AWS)	0.91	0.73	0.81	0.71	207
Google Cloud	0.81	0.70	0.76	0.62	267
Microsoft Azure	0.90	0.75	0.82	0.70	580

7.4 Knowledge Graph Demonstration & Retrieval Examples

In this section, the functionality of the artifact output by the proposed pipeline is demonstrated. The demonstration is performed through Neo4j AuraDB[15], an automated graph database that is provided as a cloud service, selected for its efficiency, reliability, and scalability. The data visualisation and querying capabilities that come with it offer a suitable environment to showcase the knowledge graph's practicality.

7.4.1 Neo4j Aura Setup

The nodes of the knowledge graph are categorized based on their roles/ types using **node labels**, making it easier to organize, query, and manage data within the graph. The node labels used in this case are shown in Figure 7.1

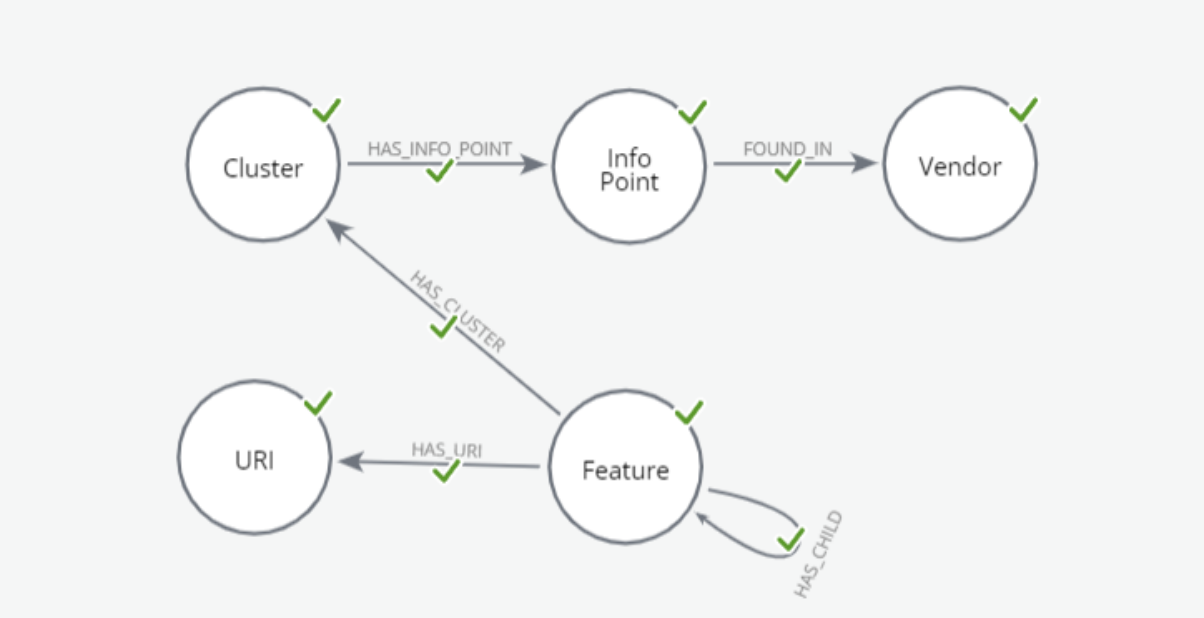

Figure 7.1: The node labels used in the knowledge graph

Each node label is associated with a CSV file, imported into the graph database and contains a row of attributed values for each instance of the defined node category. As is common practice with relational databases, these CSV files typically include a unique identifier for each node, ensuring that each entity can be uniquely identified within the graph. This unique identifier (the primary key in relational databases) is essential for establishing relationships between nodes accurately and consistently. The "building blocks" of the knowledge graph are the node instances associated with the **Info Point** node label. These represent

[15] `https://neo4j.com/product/auradb/`

the content items collected from each vendor's web pages, as described in section 5.3.3, which comprise the data points involved in clustering and subsequent feature identification through the classification process. As previously explained, more than one cluster may be associated with the same feature node, which is also the case for the collected DBPedia entries (**URI** node label). The hierarchy among feature nodes is established through the **HAS_CHILD** relationship between specific instances. The querying capability described in the following section allows for the retrieval and display of any number of node instances in the graph.

7.4.2 Graph Exploration and Retrieval

Similarly to relational databases, graph databases also support their own query languages to facilitate the efficient retrieval of information from database instances. The query language used in AuraDB is Cypher, which is similar in style to SQL but optimized for querying large, interconnected datasets, leveraging the graph structure for efficient traversal and retrieval. The following Cypher query return the central node of the graph (can also be used for any other feature node or other node labels), representing the cloud computing concept:

```
MATCH (cc:Feature {Name: "Cloud computing"})
RETURN cc;
```

Each node can be expanded to view its connections with any other node in the graph. Figure 7.2 partly displays the hierarchical structure of the graph up to the 3rd level down the hierarchy from the central node:

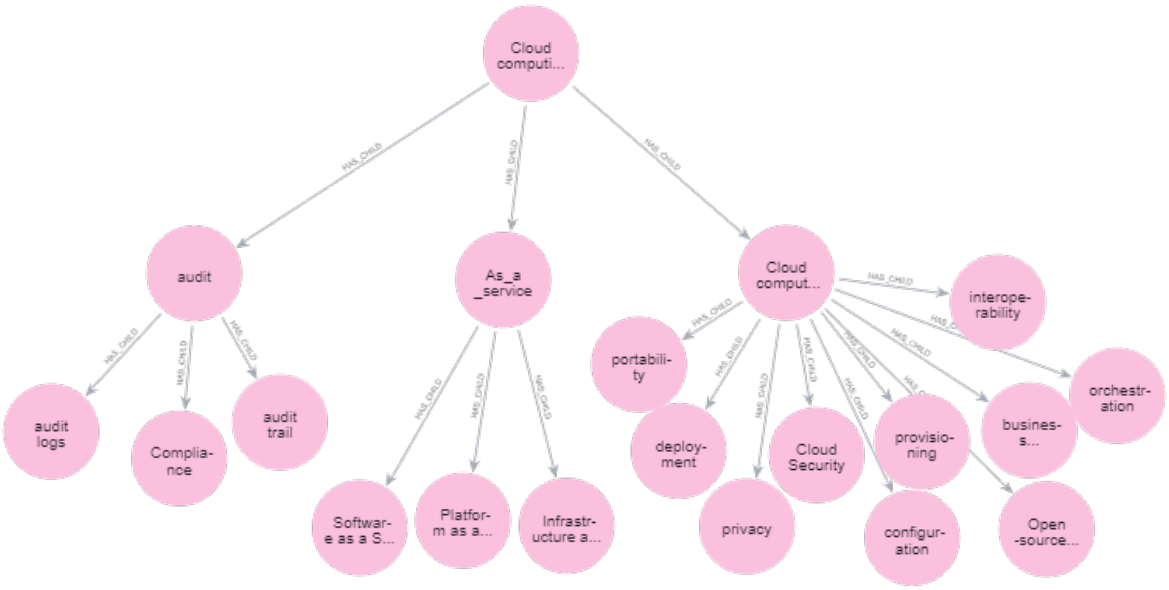

Figure 7.2: Partial view of the hierarchical structure of the graph

The nodes returned by running each query can be explored further by double-clicking to expand each node, displaying its relationship with any other node label instance found in the graph. In the following example (Figure 7.3), the node named "VMWare" is expanded to show its parent node "virtual machines" in pink, the associated URIs in orange, the clusters that have formed the node in green, and the individual mentions of that phrases from different vendors in light blue. All the properties stored for each node are accessible. Each **Info Point** node can be expanded to reveal the URL of the webpage from which it was extracted, as well as the specific content item with the text containing the term of interest (Figure 7.4).

Alternatively, the graph can be explored concerning specific vendors to display their associated feature categories based on the extracted web data. The following rather more complex Cypher query serves this exact purpose:

```
MATCH (v:Vendor)<-[:SUPPLIED_BY]-(ip:"Info Point")<-[:HAS_INFO_POINT]-(c:Cluster)<-[:HAS_CLUSTER
WHERE v.name CONTAINS "Amazon"
WITH f, v
MATCH path=(f)<-[:HAS_CHILD*]-(parent:Feature)  // Traversing up the hierarchy
RETURN f, v, path
```

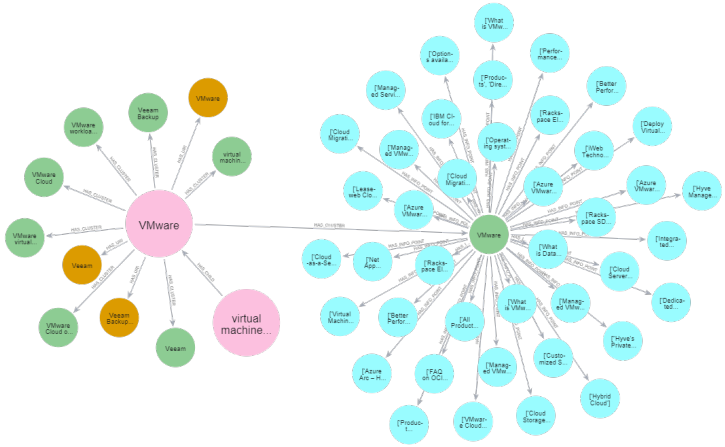

Figure 7.3: Expanded view of the "VMWare" node.

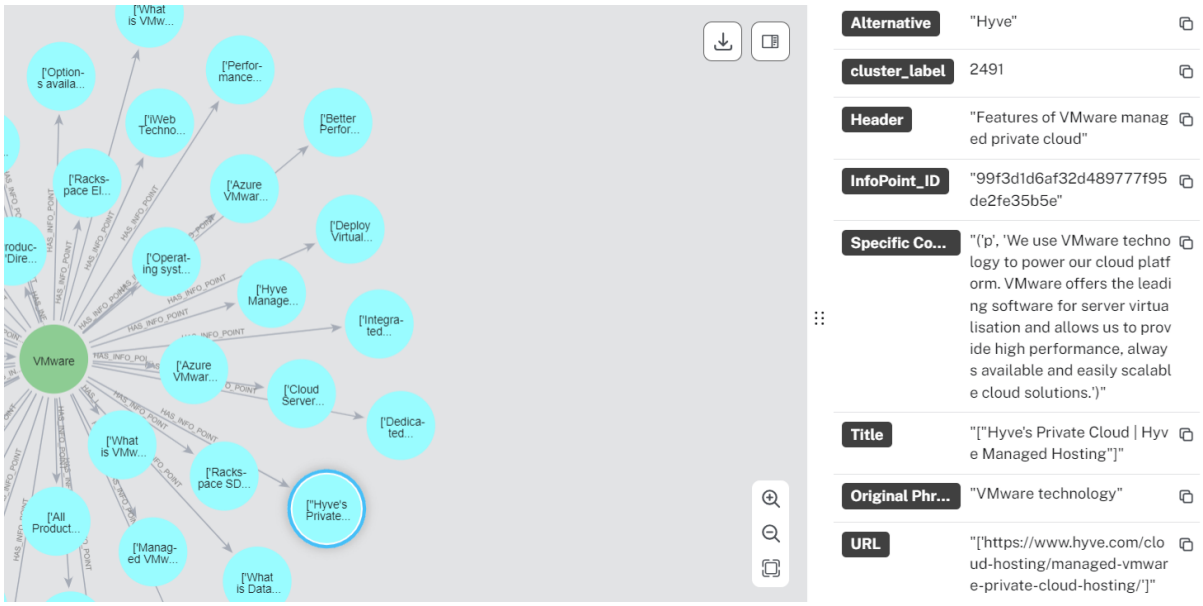

Figure 7.4: Expanded view of an **Info Point** node.

In synopsis, this query locates a vendor node by its name, computes all the feature nodes for which there is a connected path via the node labels and relationships presented in Figure 7.1, and returns the vendor along with the feature nodes satisfying this condition. In this example, the hierarchy of feature nodes is traversed up to the root node to display the broader categories of the detected features. The output is shown below in both narrow and wide views:

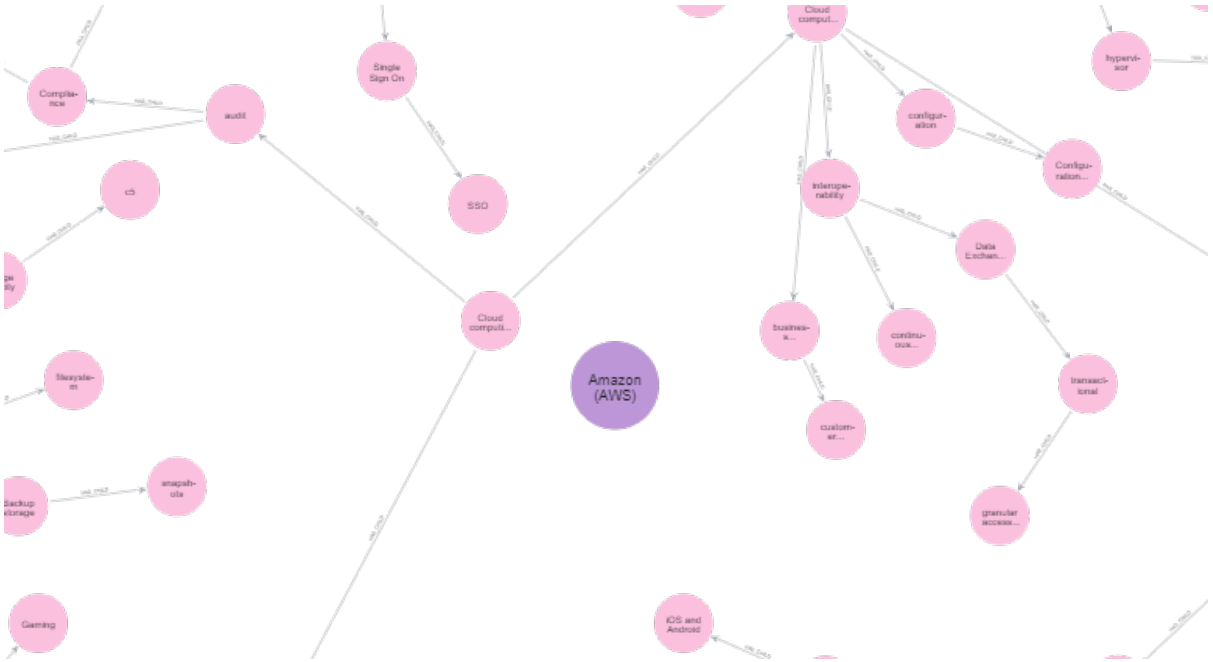

Figure 7.5: Amazon Web Services (AWS) features narrow view

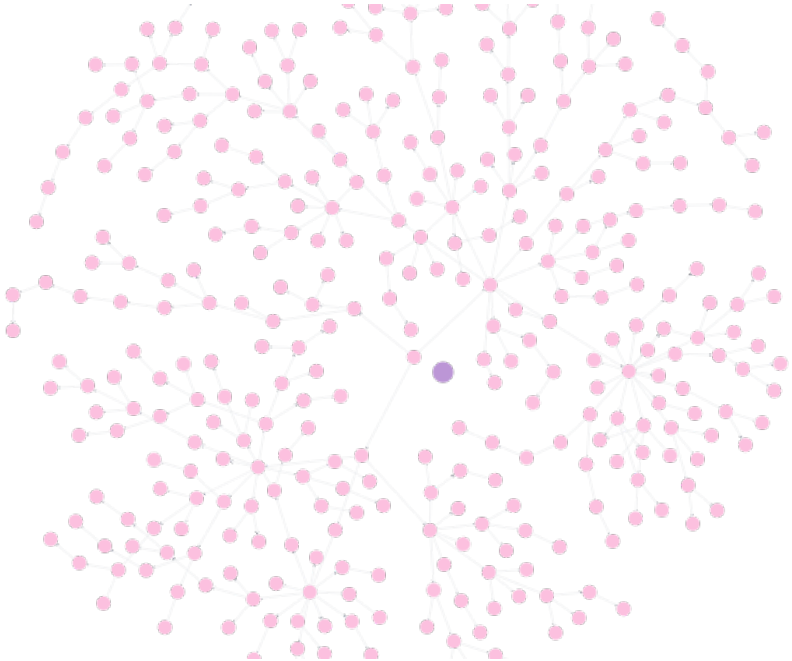

Figure 7.6: Amazon Web Services (AWS) features wide view

8 Discussion & Conclusion

The conclusion of this research aims to address the research questions outlined in section 1.4, discuss the extent to which they have been resolved, and discuss the implications and limitations of this project.

8.1 Research Objectives

Addressing the individual research sub-questions listed can provide a more comprehensive perspective on assessing the overall contribution to the main research objective.

RQ1: *Which unsupervised machine learning methods have been explored for capturing and structuring knowledge about software products from vendors' online resources into a knowledge graph?*
The systematic literature review process described in chapter 3 revealed numerous ways to tackle the knowledge extraction task. Knowledge extraction typically consists of entity extraction, attribute extraction and relation extraction (Z. Zhao et al., 2018). In existing research, entities and their associations are primarily identified using rule-based, machine-learning-based, and deep learning-based methods (L. Wang et al., 2023). Some researchers suggest software-specific NER techniques (Feng et al., 2018) while others employ conditional random field (CRF) and bi-directional long and short-term memory (BiLSTM) neural networks for named entity recognition (Nayak et al., 2020). A general observation that can be made is that most studies focus extracting software-specific entities, where a pre-defined set of categories is defined. This approach can be counter-productive in terms of deriving newly emerged features, considering the pace at which innovation in technological domains takes place and the labor-intensive nature of manual data labeling.

Unsupervised techniques such as clustering, topic modeling, Formal Concept Analysis (FCA) - (Cimiano et al., 2005), and others are usually employed to discover hidden patterns, group similar items, or identify latent topics and rising trends within large corpora of text data. However, the challenge with these approaches lies in their interpretability and conversion into structured knowledge, which often requires domain experts to convey and validate the results.

RQ2: *What are the key characteristics and capabilities of unsupervised machine learning methods used to extract knowledge from software vendors' resources?*
The literature also revealed a myriad of methods used for different natural language processing applications. In terms of data processing and representation, there are common techniques such as Part of Speech (POS) tagging, N-grams, stemming and lemmatization etc, which are essential to standardize and clean textural data, enhancing its quality for subsequent processing. Additionally, various vectorization methods are employed to transform textual data into numerical representations that machine learning algorithms can work with.

For the representation and understanding of language for more advanced tasks such as classification, there is an apparent rise in the use of language models due to the innovative exploitation of the transformer architecture and its ability to capture complex patterns, contextual relationships, and semantic meanings within text, as outlined in section 6. This allowed for the extensive unsupervised training of large language models (LLMs) on myriads of texts from different domains, enabling these models to leverage their inherent knowledge in diverse downstream tasks (Minaee et al., 2021). Nonetheless, this usually entails the need for labeled datasets to fine-tune these models for these tasks. Moreover, these models are computationally intensive, and there are concerns about their interpretability and reasoning process. Despite the challenges, the use of LLMS continuously grows as researchers and practitioners adopt strategies like transfer learning and few-shot learning to mitigate the need for large amounts of labeled data (Kaplan et al., 2020). However, limited research exists on the exploitation of LLMs in automating KG construction (Hur et al., 2021).

RQ3: *How can the effectiveness and accuracy of unsupervised machine learning methods for extracting knowledge about software products be evaluated?*
Depending on the specific task, a range of metrics can be applied to evaluate the performance of different NLP methods. The metrics that seem to stand out in the literature are classification metrics because of the dominance of this task in the design and development of NLP systems. However, these classification metrics may not be directly applicable in more complex evaluations, such as those required for a knowledge graph, where additional metrics may be utilized, such as coverage (the extent to which the graph captures the relevant domain knowledge) and path analysis (evaluating the validity and relevance of paths between entities). Expert validation is usually required to qualitatively asses the accuracy and

practicality of such artifacts.

RQ4: *How can an automated system be designed to extract and structure knowledge about software products from different vendors into a unified knowledge graph?*
The proposed architecture described in chapter 5 demonstrates the feasibility of creating systems that can automatically acquire knowledge regarding different cloud service vendors, which can be potentially generalised for use in other domains. The pipeline design was guided by the principles of the design science cycle, which proved to be highly effective when aligned with the findings from the systematic literature review, as many of the models and techniques identified in the review were instrumental during the implementation process.

RQ5: *How can the performance and quality of a system for automatically generating a knowledge graph from software vendors' data be evaluated?*
Following the previous remarks, the evaluation of the proposed system utilized different metrics to assess individual pipeline components, from the clustering quality to the classification performance and the overall coverage and information retrieval capabilities of the eventual knowledge graph. There is undoubtedly room for improvement in the evaluation process since the contribution of a cloud domain expert in the data labeling process, and the qualitative assessment of the knowledge graph would solidify the effectiveness of the system and the techniques employed by it.

8.2 Main Contribution

The analyses given above (RQ1 & RQ2) indicate the current state of the art in terms of knowledge graph construction for specific domains. As presented in (Abu-Salih, 2021), the current approaches fall under two main categories: **(1)Knowledge-based:** methods that mainly involve domain experts and human crafted rules to incorporate domain knowledge, and **(2)Learning-based:** methods that employ (un)supervised learning for entities recognition and relations extraction.

These methods limit the scope of knowledge graphs to entity-based representations, whereas the proposed approach aims to extend this by constructing knowledge graphs tailored specifically to software features. Despite the limitations mentioned above, this project shows the feasibility of automating this process. Each component of the artifact can be individually optimized further using state-of-the-art techniques for each of the tasks involved. However, the design of the proposed artifact is novel in its integration of these components into a cohesive pipeline, different from the standard approach presented in (Hofer et al., 2024). It is demonstrated that LLMs can be leveraged to construct knowledge graphs with limited intervention, providing a framework that is original in concept.

8.3 Limitations & Future Work

Following are some of the identified limitations of this study:

- Broader Applicability and Generalizability of the Artifact: One of the primary limitations of this study lies in the analysis of the artifact's broader applicability. The current design and implementation were evaluated within a specific software domain, that of cloud computing. Other software domains can vary significantly in terms of their technical terms and functionalities. This variability introduces challenges in generalizing the artifact to other domains without further adaptation. For example, differences between enterprise software, consumer applications, and specialized domains (e.g., healthcare or finance) may affect how accurately features can be extracted and interpreted. Moreover, the artifact as is utilizes an existing domain ontology as a blueprint for the output knowledge graph's structure, upon which it extends to accommodate more feature categories. Such resources may not be available in other domains and may need to be curated manually by domain experts.

- Data Diversity and Bias in Feature Extraction: Another limitation relates to data diversity and potential biases in the feature extraction process. Since the artifact relies on textual data from vendor websites, the content may exhibit biases depending on the data sources selected, such as biased terminology, varying levels of technical specificity, and marketing-driven language. Furthermore, vendor-provided information may emphasize some feature s while omitting others, leading to an incomplete or skewed representation of the software's capabilities. These biases may compromise the accuracy and completeness of extracted features, particularly when applied to software types not well-represented in the initial data. To mitigate this, future work could incorporate data from

additional sources and incorporate techniques to detect and adjust for data bias. One possible extension could be to incorporate sentiment analysis of user reviews for specific aspects of the software from each vendor (ABSA), a technique that was also discussed in section 3.7.1.

- Need for Qualitative Assessment from End-Users: Finally, a significant limitation is the absence of a qualitative assessment from end-users who would interact with the artifact in a practical setting. While quantitative metrics provide an initial understanding of the artifact's performance, they do not fully capture the usability, interpretability, or value of the extracted features from a user's perspective. Feedback from domain experts and end-users, such as software engineers, product managers, or analysts, would provide valuable insights into the artifact's practical utility, highlighting any gaps in feature relevance, ease of interpretation, or integration with existing workflows. This feedback could also help identify areas for improvement, ultimately guiding future iterations of the artifact to better meet the needs of its intended audience.

- LLM Selection: Benchmarking the proposed approach against general-purpose LLMs, such as ChatGPT, can provide means to evaluate their capacity for classification tasks in this domain. While this project utilized Gemini as the primary LLM for classification, comparing its performance with other models could provide valuable insights into the strengths and limitations of different architectures. This investigation would not only help determine the most suitable model for the task but also shed light on the adaptability of general-purpose LLMs in handling domain-specific classification challenges. Such a benchmarking study would contribute to a deeper understanding of how different models perform in knowledge extraction.

To address the aforementioned limitations, future work can be dedicated in examining specific use cases of the system through case studies and real-world scenario evaluation by practitioners and organizations that apply methods in which such a system can be utilized, such as the technology selection process. The scope of this evaluation could be extended to other product domains. The requirement for applying automatic knowledge extraction about products is the adequate availability of textual data from web sources and product listing pages for the unsupervised techniques to take effect. As mentioned in section 7.3, the data collection is crucial. Certainly, it requires further refinement to ensure the best quality possible for the input data and to produce more accurate knowledge graphs that sufficiently represent the current state of a market.

8.4 Conclusion

The main research question, reiterated below, has been investigated from different angles.

MRQ: *How can software practitioners be supported in their technology selection process with automatically updated knowledge regarding software products across different vendors?*

One of the main challenges identified through this project is the scarce availability of labeled data for training models to extract knowledge from software vendor documents. Traditionally, domain experts have been employed to manually annotate data, ensuring high accuracy and relevance. However, this approach introduces several limitations. It is prone to bias, as the data reflects the individuals involved' subjective understanding of any domain. An alternative solution to this has been explored, by utilising few-shot classification techniques instead, like few-shot prompting. While using the same individual for both the annotation and prompt engineering may limit the objectivity of the assessment of this technique, it may also enhance the likelihood that the designed prompts closely reflect the expected outcomes to the person overtaking the labeling process. This may lead to the conclusion that instead of utilizing expert personnel and resources to manually label vast amounts of data, their direct corporation, using unsupervised techniques (e.g. prompt engineering or active learning) with intently trained and informed language models can produce similar results.

Integrating LLMs with knowledge graphs may offer a scalable and adaptable solution for automating the annotation process and maintaining an up-to-date knowledge base. It becomes possible to create continuously self-updating systems that capture new trends and innovations in real-time. LLMs can handle vast amounts of data efficiently, ensuring annotation consistency and reducing the risk of subjective bias. Additionally, they can learn from new data through retraining or dynamic online learning approaches; knowledge graphs, in turn, serve as a structured foundation for organizing and verifying the knowledge generated, with human experts providing oversight to ensure its accuracy.

Future work could refine the data collection processes, further exploring use cases across different product domains. Furthermore, enhancing the integration of domain-specific ontologies and exploring more sophisticated ways to construct the domain ontology used to structure the knowledge graph are critical areas for further examination. The proposed system can be improved by addressing these topics to provide even more robust decision support tools.

Overall, this research indicates the potential to automate knowledge graph construction to support software practitioners in technology selection. The proposed system addresses the challenges of keeping pace with rapid technological changes by automating the extraction and organization of knowledge from software vendor documents and reducing the dependency on manual data annotation.

References

Abdi, H., & Williams, L. J. (2010). Principal component analysis. *WIREs Computational Statistics*, *2*(4), 433-459. Retrieved from https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.101 doi: https://doi.org/10.1002/wics.101

Abu-Salih, B. (2021). *Domain-specific knowledge graphs: A survey.* Retrieved from https://arxiv.org/abs/2011.00235

Ahmad, A., Feng, C., Khan, M., Khan, A., Ullah, A., Nazir, S., & Tahir, A. (2020, 07). A systematic literature review on using machine learning algorithms for software requirements identification on stack overflow. *Security and Communication Networks*, *2020*, 19. doi: 10.1155/2020/8830683

Akbik, A., Bergmann, T., & Vollgraf, R. (2019, June). Pooled contextualized embeddings for named entity recognition. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 724–728). Minneapolis, Minnesota: Association for Computational Linguistics. Retrieved from https://aclanthology.org/N19-1078 doi: 10.18653/v1/N19-1078

Alarte, J., & Silva, J. (2021, 06). Page-level main content extraction from heterogeneous webpages. *ACM Transactions on Knowledge Discovery from Data*, *15*, 1-105. doi: 10.1145/3451168

Alfazi, A., Sheng, Q. Z., Qin, Y., & Noor, T. H. (2015). Ontology-based automatic cloud service categorization for enhancing cloud service discovery. In *2015 ieee 19th international enterprise distributed object computing conference* (p. 151-158). doi: 10.1109/EDOC.2015.30

Amershi, S., Begel, A., Bird, C., Deline, R., Gall, H., Kamar, E., ... Zimmermann, T. (2019, 05). Software engineering for machine learning: A case study. In (p. 291-300). doi: 10.1109/ICSE-SEIP.2019.00042

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013). *Feature-oriented software product lines.* doi: 10.1007/978-3-642-37521-7

Badampudi, D., Wnuk, K., Wohlin, C., Franke, U., Smite, D., & Cicchetti, A. (2018). A decision-making process-line for selection of software asset origins and components. *Journal of Systems and Software*, *135*, 88-104. Retrieved from https://www.sciencedirect.com/science/article/pii/S0164121217302182 doi: https://doi.org/10.1016/j.jss.2017.09.033

Bagdasaryan, E., Poursaeed, O., & Shmatikov, V. (2019). Differential privacy has disparate impact on model accuracy. In *Proceedings of the 33rd international conference on neural information processing systems.* Red Hook, NY, USA: Curran Associates Inc.

Bakar, N. H., Kasirun, Z. M., & Salleh, N. (2015). Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. *Journal of Systems and Software*, *106*, 132-149. Retrieved from https://www.sciencedirect.com/science/article/pii/S0164121215001004 doi: https://doi.org/10.1016/j.jss.2015.05.006

Bakar, N. H., Kasirun, Z. M., Salleh, N., & Jalab, H. A. (2016). Extracting features from online software reviews to aid requirements reuse. *Applied Soft Computing*, *49*, 1297-1315. Retrieved from https://www.sciencedirect.com/science/article/pii/S1568494616303830 doi: https://doi.org/10.1016/j.asoc.2016.07.048

Belsis, P., Koutoumanos, A., & Sgouropoulou, C. (2013, 05). Pburc: A patterns-based, unsupervised requirements clustering framework for distributed agile software development. *Requirements Engineering*. doi: 10.1007/s00766-013-0172-9

Bergamaschi, S., Castano, S., Vincini, M., & Beneventano, D. (2001). Semantic integration of heterogeneous information sources. *Data & Knowledge Engineering*, *36*(3), 215-249. Retrieved from https://www.sciencedirect.com/science/article/pii/S0169023X00000471 (Heterogeneous Information Resources Need Semantic Access) doi: https://doi.org/10.1016/S0169-023X(00)00047-1

Bieger, V. (2023). *A decision support framework for multi-cloud service composition* (Unpublished master's thesis).

Bing, L., Wong, T.-L., & Lam, W. (2016, 04). Unsupervised extraction of popular product attributes from e-commerce web sites by considering customer reviews. *ACM Transactions on Internet Technology*, *16*, 1-17. doi: 10.1145/2857054

Bookstein, A., Kulyukin, V., & Raita, T. (2002, 10). Generalized hamming distance. *Information Retrieval*, *5*. doi: 10.1023/A:1020499411651

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). *Language models are few-shot learners.* Retrieved from `https://arxiv.org/abs/2005.14165`

Chin, K.-S., & Fu, C. (2014). Integrated evidential reasoning approach in the presence of cardinal and ordinal preferences and its applications in software selection. *Expert Systems with Applications*, *41*(15), 6718-6727. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0957417414002693` doi: https://doi.org/10.1016/j.eswa.2014.04.046

Cimiano, P., Hotho, A., & Staab, S. (2005, August). Learning concept hierarchies from text corpora using formal concept analysis. *Journal of Artificial Intelligence Research*, *24*, 305–339. Retrieved from `http://dx.doi.org/10.1613/jair.1648` doi: 10.1613/jair.1648

Constantinou, A. (2024, October). *Automatic software product features extraction from software vendor documents.* Zenodo. Retrieved from `https://doi.org/10.5281/zenodo.13899562` doi: 10.5281/zenodo.13899562

Das, M., K., S., & Alphonse, P. J. A. (2023). *A comparative study on tf-idf feature weighting method and its analysis using unstructured dataset.* Retrieved from `https://arxiv.org/abs/2308.04037`

Davril, J.-M., Delfosse, E., Hariri, N., Acher, M., Cleland-Huang, J., & Heymans, P. (2013, 08). Feature model extraction from large collections of informal product descriptions. *ESEC/FSE*. doi: 10.1145/2491411.2491455

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *Bert: Pre-training of deep bidirectional transformers for language understanding.* Retrieved from `https://arxiv.org/abs/1810.04805`

Du, D., Ren, X., Wu, Y., Chen, J., Ye, W., Sun, J., ... Zhang, S. (2018). Refining traceability links between vulnerability and software component in a vulnerability knowledge graph. In T. Mikkonen, R. Klamma, & J. Hernández (Eds.), *Web engineering* (pp. 33–49). Cham: Springer International Publishing.

Du, T. C., Li, F., & King, I. (2009). Managing knowledge on the web – extracting ontology from html web. *Decision Support Systems*, *47*(4), 319-331. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0167923609000542` (Smart Business Networks: Concepts and Empirical Evidence) doi: https://doi.org/10.1016/j.dss.2009.02.011

Farshidi, S. (2020). *Multi-criteria decision-making in software production* (Unpublished doctoral dissertation).

Farshidi, S., Jansen, S., & van der Werf, J. M. (2020). Capturing software architecture knowledge for pattern-driven design. *Journal of Systems and Software*, *169*, 110714. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0164121220301552` doi: https://doi.org/10.1016/j.jss.2020.110714

Farshidi, S., Rezaee, K., Mazaheri, S., Rahimi, A. H., Dadashzadeh, A., Ziabakhsh, M., ... Jansen, S. (2024). Understanding user intent modeling for conversational recommender systems: a systematic literature review. *User Modeling and User-Adapted Interaction*, 1–64.

Farshidi, S., & Zhao, Z. (2022, 05). An adaptable indexing pipeline for enriching meta information of datasets from heterogeneous repositories. In (p. 472-484). doi: 10.1007/978-3-031-05936-0_37

Feng, X., Li, Q., Wang, H., & Sun, L. (2018, 08). Acquisitional rule-based engine for discovering internet-of-thing devices..

Ferrara, A., Nikolov, A., & Scharffe, F. (2011). Data linking for the semantic web. *Int. J. Semantic Web Inf. Syst.*, *7*, 46-76. Retrieved from `https://api.semanticscholar.org/CorpusID:15886037`

Fitzgerald, B., & Stol, K.-J. (2014, 06). Continuous software engineering and beyond: Trends and challenges.. doi: 10.1145/2593812.2593813

F. Wang, B. L., J.P. Liu. (2019). Survey on construction of code knowledge graph and intelligent software development. *Journal of Software*, *31*(1), 47–66.

Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., & Smith, N. A. (2020). *Don't stop pretraining: Adapt language models to domains and tasks.* Retrieved from `https://arxiv.org/abs/2004.10964`

Haefliger, S., von Krogh, G., & Spaeth, S. (2008). Code reuse in open source software. *Management Science*, *54*(1), 180–193. Retrieved 2023-10-16, from `http://www.jstor.org/stable/20122369`

Haris, M. S., Kurniawan, T., & Ramdani, F. (2020, 12). Automated features extraction from software requirements specification (srs) documents as the basis of software product line (spl) engineering. *Journal of Information Technology and Computer Science*, *5*, 279. doi: 10.25126/jitecs.202053219

Hevner, A. R. (2002). Design science research in information systems.. Retrieved from `https://api.semanticscholar.org/CorpusID:10086156`

Hofer, M., Obraczka, D., Saeedi, A., Köpcke, H., & Rahm, E. (2024, August). Construction of knowledge graphs: Current state and challenges. *Information*, *15*(8), 509. Retrieved from `http://dx.doi.org/10.3390/info15080509` doi: 10.3390/info15080509

Hua, Y. C., Denny, P., Wicker, J., & Taskova, K. (2024, September). A systematic review of aspect-based sentiment analysis: domains, methods, and trends. *Artificial Intelligence Review*, *57*(11). Retrieved from `http://dx.doi.org/10.1007/s10462-024-10906-z` doi: 10.1007/s10462-024-10906-z

Huang, Z., Xu, W., & Yu, K. (2015). *Bidirectional lstm-crf models for sequence tagging.* Retrieved from `https://arxiv.org/abs/1508.01991`

Hur, A., Janjua, N., & Ahmed, M. (2021). *A survey on state-of-the-art techniques for knowledge graphs construction and challenges ahead.* Retrieved from `https://arxiv.org/abs/2110.08012`

Izadi, M., Ganji, S., Heydarnoori, A., & Gousios, G. (2020, 10). *Topic recommendation for software repositories using multi-label classification algorithms.*

Jadhav, A. S., & Sonar, R. M. (2011). Framework for evaluation and selection of the software packages: A hybrid knowledge based system approach. *Journal of Systems and Software*, *84*(8), 1394-1407. Retrieved from `https://www.sciencedirect.com/science/article/pii/S016412121100077X` doi: https://doi.org/10.1016/j.jss.2011.03.034

Kang, K., Cohen, S., Hess, J., Novak, W., & Peterson, A. (1990, 01). Feature-oriented domain analysis (foda) feasibility study.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... Amodei, D. (2020). *Scaling laws for neural language models.* Retrieved from `https://arxiv.org/abs/2001.08361`

Kertkeidkachorn, N., & Ichise, R. (2017). T2kg: An end-to-end system for creating knowledge graph from unstructured text. In *Aaai workshops.* Retrieved from `https://api.semanticscholar.org/CorpusID:53279967`

Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), 307–392. Retrieved from `http://dx.doi.org/10.1561/2200000056` doi: 10.1561/2200000056

Kitchenham, B. (2004, 08). Procedures for performing systematic reviews. *Keele, UK, Keele Univ.*, *33*.

Kuwajima, H., Yasuoka, H., & Nakae, T. (2020, 05). Engineering problems in machine learning systems. *Machine Learning*, *109*. doi: 10.1007/s10994-020-05872-w

Lemos, O. A. L., de Paula, A. C., Zanichelli, F. C., & Lopes, C. V. (2014). Thesaurus-based automatic query expansion for interface-driven code search. In *Proceedings of the 11th working conference on mining software repositories* (p. 212–221). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2597073.2597087` doi: 10.1145/2597073.2597087

Li, H., Li, S., Sun, J., Xing, Z., Peng, X., Liu, M., & Zhao, X. (2018). Improving api caveats accessibility by mining api caveats knowledge graph. In *2018 ieee international conference on software maintenance and evolution (icsme)* (p. 183-193). doi: 10.1109/ICSME.2018.00028

Li, H., & Lu, W. (2021). *Mixed cross entropy loss for neural machine translation.* Retrieved from `https://arxiv.org/abs/2106.15880`

Li, Y., Schulze, S., & Saake, G. (2017, 09). Reverse engineering variability from natural language documents: A systematic literature review. In (p. 133-142). doi: 10.1145/3106195.3106207

Liang, F., Hou, F., Farshidi, S., Jansen, S., et al. (2023). Sentiment analysis for software quality assessment. In *Ceur workshop proceedings* (Vol. 3567, pp. 17–24).

Liu, S., Li, Y., & Fan, B. (2018, 01). Hierarchical rnn for few-shot information extraction learning: 4th international conference of pioneering computer scientists, engineers and educators, icpcsee 2018, zhengzhou, china, september 21-23, 2018, proceedings, part ii. In (p. 227-239). doi: 10.1007/978-981-13-2206-8_20

Lizarralde, I., Mateos, C., Zunino, A., Majchrzak, T. A., & Grønli, T.-M. (2020). Discovering web services in social web service repositories using deep variational autoencoders. *Information Processing & Management*, *57*(4), 102231. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0306457319310878` doi: https://doi.org/10.1016/j.ipm.2020.102231

Ma, D., & Kauffman, R. (2014, 11). Competition between software-as-a-service vendors. *IEEE Transactions on Engineering Management*, *61*, 717-729. doi: 10.1109/TEM.2014.2332633

Mao, A., Mohri, M., & Zhong, Y. (2023). *Cross-entropy loss functions: Theoretical analysis and applications.* Retrieved from `https://arxiv.org/abs/2304.07288`

McInnes, L., Healy, J., & Astels, S. (2017, 03). hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, *2*. doi: 10.21105/joss.00205

Meijer, K., Frasincar, F., & Hogenboom, F. (2014a). A semantic approach for extracting domain taxonomies from text. *Decision Support Systems*, *62*, 78-93. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0167923614001031` doi: https://doi.org/10.1016/j.dss.2014.03.006

Meijer, K., Frasincar, F., & Hogenboom, F. (2014b). A semantic approach for extracting domain taxonomies from text. *Decision Support Systems*, *62*, 78-93. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0167923614001031` doi: https://doi.org/10.1016/j.dss.2014.03.006

Miguel, J., Mauricio, D., & Rodriguez, G. (2014, 11). A review of software quality models for the evaluation of software products. *International journal of Software Engineering & Applications*, *5*, 31-54. doi: 10.5121/ijsea.2014.5603

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). *Efficient estimation of word representations in vector space.* Retrieved from `https://arxiv.org/abs/1301.3781`

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013b, 01). Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, *2013*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality.* Retrieved from `https://arxiv.org/abs/1310.4546`

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2021, apr). Deep learning–based text classification: A comprehensive review. *ACM Comput. Surv.*, *54*(3). Retrieved from `https://doi.org/10.1145/3439726` doi: 10.1145/3439726

Minka, T. (2000). Automatic choice of dimensionality for pca. In T. Leen, T. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems* (Vol. 13). MIT Press. Retrieved from `https://proceedings.neurips.cc/paper_files/paper/2000/file/7503cfacd12053d309b6bed5c89de212-Paper.pdf`

Nayak, A., Kesri, V., & Dubey, R. (2020, 01). Knowledge graph based automated generation of test cases in software engineering. In (p. 289-295). doi: 10.1145/3371158.3371202

Pekar, V., & Staab, S. (2002). Taxonomy learning - factoring the structure of a taxonomy into a semantic classification decision. In *COLING 2002: The 19th international conference on computational linguistics.* Retrieved from https://aclanthology.org/C02-1090

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Conference on empirical methods in natural language processing.* Retrieved from https://api.semanticscholar.org/CorpusID:1957433

Pimpalkar, A., & Raj R, J. R. (2022). Mbilstmglove: Embedding glove knowledge into the corpus using multi-layer bilstm deep learning model for social media sentiment analysis. *Expert Systems with Applications*, *203*, 117581. Retrieved from https://www.sciencedirect.com/science/article/pii/S0957417422008946 doi: https://doi.org/10.1016/j.eswa.2022.117581

Pujol, D., McKenna, R., Kuppam, S., Hay, M., Machanavajjhala, A., & Miklau, G. (2020). Fair decision making using privacy-protected data. In *Proceedings of the 2020 conference on fairness, accountability, and transparency* (p. 189–199). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3351095.3372872 doi: 10.1145/3351095.3372872

Qin, S., & Chow, K. P. (2019). Automatic analysis and reasoning based on vulnerability knowledge graph. In H. Ning (Ed.), *Cyberspace data and intelligence, and cyber-living, syndrome, and health* (pp. 3–19). Singapore: Springer Singapore.

Quirchmayr, T., Paech, B., Kohl, R., & Karey, H. (2017, 02). Semi-automatic software feature-relevant information extraction from natural language user manuals. In (Vol. 10153, p. 255-272). doi: 10.1007/978-3-319-54045-0_19

Rani, A., Mishra, D., & Omerovic, A. (2022). Multi-vendor software ecosystem: Challenges from company' perspective. In A. Rocha, H. Adeli, G. Dzemyda, & F. Moreira (Eds.), *Information systems and technologies* (pp. 382–393). Cham: Springer International Publishing.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (p. 1135–1144). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2939672.2939778 doi: 10.1145/2939672.2939778

Rokach, L., & Maimon, O. (2005). Decision trees. In O. Maimon & L. Rokach (Eds.), *Data mining and knowledge discovery handbook* (pp. 165–192). Boston, MA: Springer US. Retrieved from https://doi.org/10.1007/0-387-25465-X_9 doi: 10.1007/0-387-25465-X_9

Rus, I., Halling, M., & Biffl, S. (2003, 10). Supporting decision-making in software engineering with process simulation and empirical studies. *International Journal of Software Engineering and Knowledge Engineering*, *13*, 531-545. doi: 10.1142/S0218194003001391

Sarawagi, S. (2008, 01). Information extraction. *Foundations and Trends in Databases*, *1*, 261-377. doi: 10.1561/1500000003

Sarker, I. (2021, 03). Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, *2*. doi: 10.1007/s42979-021-00592-x

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85-117. Retrieved from https://www.sciencedirect.com/science/article/pii/S0893608014002135 doi: https://doi.org/10.1016/j.neunet.2014.09.003

Schröder, C., & Niekler, A. (2020). *A survey of active learning for text classification using deep neural networks.* Retrieved from https://arxiv.org/abs/2008.07267

Shang, W., & Huang, X. (2024). *A survey of large language models on generative graph analytics: Query, learning, and applications.* Retrieved from https://arxiv.org/abs/2404.14809

Sommerville, I. (1985). *Software engineering (2nd ed.).* USA: Addison-Wesley Longman Publishing Co., Inc.

Sutton, C., & McCallum, A. (2010). *An introduction to conditional random fields.* Retrieved from https://arxiv.org/abs/1011.4088

Tam, K. Y. (1993). Applying conceptual clustering to knowledge-bases construction. *Decision Support Systems*, *10*(2), 173-198. Retrieved from https://www.sciencedirect.com/science/article/pii/0167923693900374 doi: https://doi.org/10.1016/0167-9236(93)90037-4

Wang, J., Zhang, X., & Chen, L. (2021). How well do pre-trained contextual language representations recommend labels for github issues? *Knowledge-Based Systems*, *232*, 107476. Retrieved from https://www.sciencedirect.com/science/article/pii/S0950705121007383 doi: https://doi.org/10.1016/j.knosys.2021.107476

Wang, L., Sun, C., Zhang, C., Nie, W., & Huang, K. (2023). Application of knowledge graph in software engineering field: A systematic literature review. *Information and Software Technology*, *164*, 107327. Retrieved from https://www.sciencedirect.com/science/article/pii/S0950584923001829 doi: https://doi.org/10.1016/j.infsof.2023.107327

Wang, Y., Yao, Q., Kwok, J., & Ni, L. M. (2020). *Generalizing from a few examples: A survey on few-shot learning.* Retrieved from https://arxiv.org/abs/1904.05046

Wei, C.-C., & Wang, M.-J. J. (2004). A comprehensive framework for selecting an erp system. *International Journal of Project Management*, *22*(2), 161-169. Retrieved from https://www.sciencedirect.com/science/article/pii/S0263786302000649 doi: https://doi.org/10.1016/S0263-7863(02)00064-9

Weikum, G., Dong, L., Razniewski, S., & Suchanek, F. (2021). *Machine knowledge: Creation and curation of comprehensive knowledge bases.* Retrieved from https://arxiv.org/abs/2009.11564

Wenpeng, L., Jianbin, W., Zeqi, L., Junfeng, Z., Yanzhen, Z., & Bing, X. (2017). Software knowledge graph building method for open source project. *Journal of Frontiers of Computer Science & Technology*, *11*(6), 851.

Wong, T.-L., Lam, W., & Wong, T.-S. (2008, 07). An unsupervised framework for extracting and normalizing product attributes from multiple web sites. In (p. 35-42). doi: 10.1145/1390334.1390343

Xin, W., Liu, X., Liu, J., Chen, X., & Wu, H. (2021, 05). A novel knowledge graph embedding based api recommendation method for mashup development. *World Wide Web*, *24*. doi: 10.1007/s11280-021-00894-3

Xu, D., & jie Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science*, *2*, 165 - 193. Retrieved from https://api.semanticscholar.org/CorpusID:54134680

Zhang, J., Xie, R., Ye, W., Zhang, Y., & Zhang, S. (2020). Exploiting code knowledge graph for bug localization via bi-directional attention. In *2020 ieee/acm 28th international conference on program comprehension (icpc)* (p. 219-229). doi: 10.1145/3387904.3389281

Zhang, N., Wang, J., He, K., Li, Z., & Huang, Y. (2019, 03). Mining and clustering service goals for restful service discovery. *Knowledge and Information Systems*, *58*. doi: 10.1007/s10115-018-1171-4

Zhao, X., Xing, Z., Kabir, M. A., Sawada, N., Li, J., & Lin, S.-W. (2017). Hdskg: Harvesting domain specific knowledge graph from content of webpages. In *2017 ieee 24th international conference on software analysis, evolution and reengineering (saner)* (p. 56-67). doi: 10.1109/SANER.2017.7884609

Zhao, Z., Han, S.-K., & So, I.-M. (2018). Architecture of knowledge graph construction techniques.. Retrieved from https://api.semanticscholar.org/CorpusID:207900787

Zhou, C., Li, B., Sun, X., & Guo, H. (2018). Recognizing software bug-specific named entity in software bug repository. In *Proceedings of the 26th conference on program comprehension* (p. 108–119). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3196321.3196335 doi: 10.1145/3196321.3196335

Zhou, J., Chen, W., Wu, G., & Wei, J. (2019, 06). Semitagrec: A semi-supervised learning based tag recommendation approach for docker repositories. In (p. 132-148). doi: 10.1007/978-3-030-22888-0_10

Zhou, X., Jin, Y., Zhang, H., Li, S., & Huang, X. (2016). A map of threats to validity of systematic literature reviews in software engineering. In *2016 23rd asia-pacific software engineering conference (apsec)* (p. 153-160). doi: 10.1109/APSEC.2016.031

Appendices

A Model Definitions

Name	Definition
Adaptive Moment Optimization (Adam)	an extension of the stochastic gradient descent (SGD) optimization algorithm. It adaptively adjusts the learning rate for each parameter based on the estimates of both the first-order moments (mean) and second-order moments (variance) of the gradients.
Association Rule Mining	a data mining technique that aims to discover interesting relationships, associations, and dependencies between items in large datasets. It involves identifying frequently occurring itemsets and deriving association rules that express relationships between items based on their co-occurrence patterns.
Bidirectional Encoder Representations from Transformers (BERT)	an NLP model that was introduced by Google in 2018. BERT is based on the Transformer architecture, which is a deep learning model architecture that utilizes self-attention mechanisms to capture relationships between words in a sequence. BERT is pre-trained on massive amounts of text data, learning to predict missing words in sentences and can be fine-tuned for specific NLP tasks like text classification, named entity recognition, and more with relatively small amounts of task-specific data.
Bidirectional Long Short-Term Memory (BiLSTM)	a type of recurrent neural network (RNN) architecture that is capable of processing sequential data in both forward and backward directions. Widely used in NLP tasks, such as text classification, named entity recognition, sentiment analysis, and machine translation. The bidirectional processing helps the model to better understand the dependencies and patterns in the sequence, leading to improved performance in tasks that require capturing long-range dependencies.
BIO tagging	a labeling scheme commonly used in named entity recognition (NER) tasks. Used to annotate words or tokens in a sequence with labels that indicate the boundaries: (B)eginning, (I)nside, (O)utside of a named entity, followed by its type.
Co-occurence Statistical Information (CSI)	a measurement of the frequency and patterns of co-occurrence between elements or events in a dataset. Frequently used in text analysis and NLP to analyze the occurrence patterns of words, phrases, or concepts in a corpus of text, helping to uncover associations, collocations, semantic relationships, and contextual dependencies.
Conditional Random Field (CRF)	a probabilistic graphical model used for structured prediction tasks, particularly in sequence labeling problems such as part-of-speech tagging, named entity recognition, and syntactic parsing. Training often involves optimization algorithms such as maximum likelihood estimation or gradient-based methods.
Continuous Bag of Words (CBOW)	a shallow neural network model that aims to learn distributed representations (word embeddings) of words based on their context within a given window of surrounding words. CBOW is particularly useful for generating word embeddings when the contextual information surrounding a word is important. It is computationally efficient compared to other models such as RNNs or tranformers, however, it may not capture long-range dependencies as effectively.
Convolutional Neural Network (CNN)	a type of deep learning neural network architecture that is particularly effective for processing grid-like data, such as images, videos, and sequential data like time series or text. The ability of CNNs to automatically learn hierarchical representations of data and capture local and spatial dependencies makes them powerful tools for extracting meaningful features from complex data.
Cosine Similarity	a measure of similarity between two vectors in a vector space. It is commonly used in various domains, including NLP, information retrieval, and recommendation systems, to compare the similarity between documents, sentences, or any other objects represented as vectors. Ranges between -1 and 1, where values close to 1 indicate high similarity, values close to 0 indicate no strong similarity or dissimilarity, and values close to -1 indicate high dissimilarity.
Cross Entropy Loss	a commonly used loss function in machine learning, particularly in classification tasks. It measures the dissimilarity between predicted probability distributions and true probability distributions.
Decision Tree	a supervised machine learning algorithm used for both classification and regression tasks. It is a flowchart-like model that makes decisions based on the features or attributes of the input data. The decision tree algorithm creates a tree-like structure where each internal node represents a test on a feature, each branch corresponds to an outcome of the test, and each leaf node represents a class label or a numerical value.
Deep Belief Network (DBN)	a type of generative deep learning model composed of multiple layers of restricted Boltzmann machines (RBMs). It is designed to learn hierarchical representations of data in an unsupervised manner and has been widely used in NLP and recommendation systems. Typically trained in a layer-wise, bottom-up manner where each layer is pretrained as an RBM, which is an unsupervised generative model that learns to reconstruct the input data and capture underlying dependencies.
Deep Neural Network (DNN)	a type of artificial neural network that consists of multiple layers of interconnected nodes or neurons. It is designed to learn and represent complex patterns and features of data by successively transforming the input through multiple layers of nonlinear transformations. DNNs are typically trained using gradient-based optimization methods, like backpropagation, to minimize the difference between predicted and actual outputs and require a large amount of data and significant computational resources.
Dot-Product	(also known as the scalar product) an operation that takes two equal-length sequences of numbers (usually coordinate vectors) and returns a single number. The operation is performed by multiplying corresponding entries and then summing those products. In a geometric context, the dot product of two vectors captures the cosine of the angle between them and their magnitudes, and it's a key operation in linear algebra and vector calculus.
Euclidean Distance	a measure of the straight-line distance between two points in Euclidean space. While it is widely used in numerical and geometric contexts, its direct application in NLP is limited due to the discrete and high-dimensional nature of textual data. However, Euclidean distance can still be utilized in specific scenarios in NLP, particularly when working with numerical or vector representations of text.
Fully Connected Network (FCN)	a neural network that consists of layers of fully connected (dense) nodes where each node in one layer is connected to every node in the next layer. These networks process text data in fixed-length vector representation form that serve as input to the fully connected layers, which learn patterns and relationships between the input features and the task-specific output.
Fuzzy C-Means (FCM)	an extension of the K-means clustering algorithm that incorporates fuzzy logic principles. FCM is used for partitioning data into clusters with soft boundaries, allowing data points to belong to multiple clusters to varying degrees of membership.
Gated Recurrent Unit (GRU)	a type of recurrent neural network (RNN) architecture designed to capture long-term dependencies and patterns in sequential data. It is an extension of the traditional RNN model that addresses the vanishing gradient problem and allows for more effective modeling of sequential data.
Gibbs sampling	an algorithm used for approximate inference and sampling from complex probability distributions.
Global Vectors for Word Representation (GloVe)	a word embedding model used in NLP tasks. It is designed to learn dense vector representations (embeddings) for words based on their co-occurrence statistics in a large corpus of text data.
Graph Convolutional Network (GCN)	a type of neural network designed to process data represented as graphs. The key idea of GCNs is to define a localized convolution operation on each node, taking into account the information from its neighbors in the graph. This enables GCNs to learn node representations that capture both the node's attributes and the information from its neighborhood.
Graph Neural Network (GNN)	a type of network that uses a graph structure where nodes represent entities and edges represent the relationships between these entities. It allows the processing of data structured in graphs, enabling the capture of complex patterns within the data. It's extensively used for social network analysis, molecule structure analysis, recommendation systems, and many other applications.
Hamming Distance	a metric used to measure the dissimilarity between two binary strings or vectors of equal length. In the context of label vectors in machine learning or classification tasks, the Hamming distance can be used to assess the disagreement or dissimilarity between the predicted labels and the true labels.
Jaccard Similarity	a measure used to determine the similarity or overlap between two sets. It is often employed in information retrieval, data mining, and text analysis tasks to measure the similarity between two sets of words, such as words in two documents. The Jaccard Similarity is defined as the size of the intersection of the sets divided by the size of their union. It is calculated using the following formula: J(A, B) = |A ∩ B| / |A ∪ B|
K Nearest Neighbours (KNN)	a supervised classification algorithm used for making predictions based on the similarities between feature vectors. It assigns a class label to a data point based on the majority vote of its K nearest neighbors in the feature space.
K-means	an unsupervised machine learning algorithm used for clustering and partitioning data points into distinct groups or clusters based on their similarity. It aims to minimize the intra-cluster variance and maximize the inter-cluster variance. Centroids of each cluster are iteratively recalculated by taking the mean of all data points assigned to that cluster. This moves the centroids to the center of their respective clusters.
Knowledge Graph	a structured representation of knowledge that captures entities, their attributes, and the relationships between them. Capturing and representing semantic knowledge to enhance language understanding and reasoning.
Latent Dirichlet Allocation (LDA)	a generative statistical model used for topic modeling, a technique that uncovers latent topics in a collection of documents. LDA assumes that documents are probabilistic mixtures of topics, and topics are distributions over words.
Latent Semantic Analysis (LSA)	a technique used in NLP and information retrieval to analyze and extract the underlying latent semantic structure of a collection of texts. LSA applies a mathematical method called Singular Value Decomposition (SVD) to a term-document matrix, and by reducing the dimensionality of the matrix, LSA uncovers hidden patterns and associations between terms and documents, allowing for tasks such as document similarity, information retrieval, and text classification.
Logistic Regression	a statistical model for binary classification. It is a variant of linear regression that predicts the probability of an instance belonging to a particular class based on its features. Logistic Regression models the relationship between the input features and the binary outcome using the logistic function (also known as the sigmoid function).
Naive Bayes	a probabilistic classifier based on Bayes' theorem with an assumption of independence between features. It is widely used in various machine learning and NLP tasks, particularly for text classification and sentiment analysis. Naive Bayes assumes that the features used for classification are conditionally independent of each other, given the class label.
Ontology	a formal representation of knowledge in a specific domain or a structured framework that defines the concepts, entities, relationships, and properties within that domain. In the context of NLP, ontologies are used to capture and model the semantic relationships between words, entities, and concepts in text. They serve as a knowledge base or a domain-specific vocabulary that helps in understanding and analyzing natural language data.
Ontology-based Semantic Similarity Calculation (OSSC)	a process of determining the degree of similarity between two entities (words, phrases, concepts) based on their semantic meanings and relationships in an ontology.
PageRank	an algorithm developed by Larry Page and Sergey Brin, assigns a numerical score to web pages based on their importance within a network of interlinked pages, such as the World Wide Web. The algorithm considers incoming links from other pages as votes of confidence, suggesting that pages with more high-quality incoming links are more important.
Pearson Correlation Coefficient (PCC)	a measure of the linear relationship between two variables. It quantifies the strength and direction of the linear association between two continuous variables.
Personalised PageRank (PPR)	a variant of PageRank, assigns a biased jump probability to each node and ranks graph nodes based on the graph structure. Therefore, it is able to reflect the keyterm frequency by assigning the jump probability according to the keyterm frequency.
Pointwise Mutual Information (PMI)	a statistical measure used to quantify the association between two discrete random variables or events. PMI is often employed in NLP and information retrieval tasks to determine the strength of the relationship between words or terms in a corpus. PMI(x, y) = log(P(x, y) / (P(x) * P(y))) where P(x, y) is the joint probability of events x and y co-occurring, and P(x) and P(y) are their individual probabilities.
Principal Component Analysis (PCA)	a dimensionality reduction technique commonly used in data analysis and machine learning. It aims to transform a high-dimensional dataset into a lower-dimensional space while retaining as much of the original information as possible. PCA achieves this by identifying the principal components, which are linear combinations of the original features that capture the most significant variability in the data.
Random Forest	a supervised machine learning algorithm that combines the predictions of multiple decision trees to make more accurate predictions. It is widely used for classification and regression tasks due to its simplicity, robustness, and ability to handle high-dimensional data.
Recurrent Neural Network (RNN)	a type of artificial neural network designed to process sequential and temporal data. RNNs have a unique architecture that allows them to retain and utilize information from previous time steps, making them effective in tasks involving sequential data such as NLP, speech recognition, and time series analysis.

Ridge Regularization (L2)	a regularization technique that adds a penalty proportional to the squared magnitudes of the model's parameter weights. It encourages smaller but non-zero values for all parameter weights and is effective in reducing the impact of large weight values and makes the model more robust to noise in the data.
Robustly Optimized BERT Approach (RoBERTa)	a language representation model based on the Transformer architecture. Unlike BERT, RoBERTa is pretrained using only the left-to-right (or right-to-left) context, rather than a bidirectional approach, to better leverage the context and avoid the mismatch in pretraining and fine-tuning settings. RoBERTa benefits from larger-scale training compared to BERT, using more data and training for more iterations to capture more diverse and robust representations of language.
Rule-based Algorithm	a type of algorithm that makes decisions based on a set of predefined rules or heuristics. These rules are often expressed in the form of if-then statements. For instance, an email spam filter might be rule-based and could use rules like: "If an email contains the words 'prize' and 'free' and 'click here', then mark it as spam."
Silhouette Coefficient	a metric used to evaluate the quality of clustering results. It measures how well each sample fits within its assigned cluster and how distinct the clusters are from each other. The Silhouette Coefficient ranges from -1 to 1, with higher values indicating better clustering quality.
Singular Value Decomposition (SVD)	a matrix factorization technique widely used in linear algebra and data analysis. SVD decomposes a matrix into three separate matrices, which allows for dimensionality reduction, noise reduction, and the extraction of underlying patterns or structures in the data.
Skip-gram	an algorithm used in NLP and word embedding techniques, similar to Continuous Bag of Words (CBOW). It is a shallow neural network model that aims to learn distributed representations (word embeddings) of words by predicting the context words given a target word.
Support Vector Machine (SVM)	a supervised machine learning algorithm used for classification and regression tasks. SVMs are known for their ability to handle complex datasets, especially those with clear margin-based separation. SVMs use a subset of the training data called support vectors. These are the data points closest to the decision boundary and have the most influence on the hyperplane. SVMs can handle non-linearities through the kernel trick.
Term Frequency-Inverse Document Frequency (TF-IDF)	a statistical measure used to evaluate how important a word is to a document in a corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.
TextRank	an algorithm for keyword extraction and text summarization. It is based on the concept of graph-based ranking and is inspired by Google's PageRank algorithm. TextRank represents a document as a graph and assigns importance scores to words or phrases based on their connectivity and centrality within the graph.
Word2Vec	a neural network-based technique for generating word embeddings, which are dense vector representations of words in a high-dimensional space.

B Features

Name	Definition
adjacency matrix	adjacency matrix is a square matrix that represents the relationships between elements in a graph. In the context of NLP, the graph typically represents linguistic entities (e.g., words, sentences, documents) and their relationships, such as co-occurrence or semantic connections. The elements of the adjacency matrix indicate the presence or strength of connections between the nodes in the graph.
aspect-sensitive word embeddings	specialized word embeddings that capture the aspect-specific meanings and semantic relationships of words in the context of a specific aspect or domain
attention	attention mechanisms are designed to capture the dependencies between different parts of a sequence, and multi-head attention extends this idea by applying attention multiple times in parallel.
character embeddings	unlike traditional word embeddings that represent words as dense vectors, character embeddings represent words as sequences of characters or subword units. The character embeddings capture the morphological and subword-level information of words, allowing the model to learn representations for words that are not seen in the training data.
chunking	dividing a text into syntactically correlated parts of words, like noun phrases, verb phrases, or adjective phrases. It's essentially a way of extracting short phrases from the text without needing to analyze the complete sentence structure.
classification	categorizing data points into predefined classes or categories based on their features or characteristics. In the context of NLP, classification typically involves assigning labels or categories to text documents, sentences, or individual words.
clustering	technique used to group similar data points together based on their inherent patterns or similarities
co-reference resolution	identifying and linking mentions of the same entity or concept within a text. It aims to resolve ambiguous pronouns or noun phrases by determining which other phrases in the text they refer to.
contextual representation	representation of words or phrases in a manner that takes into account their context, or the words and phrases that surround them
dimensionality reduction	process of reducing the number of features (dimensions) in the data while preserving as much relevant information as possible. NLP datasets often have a high-dimensional feature space due to the large vocabulary and sparse representations of text.
domain information	knowledge, concepts, and terminology specific to a particular subject area or industry
ensemble learning	machine learning technique that combines the predictions of multiple individual models to make more accurate and robust predictions. It involves creating an ensemble or a group of models that work together to improve overall predictive performance. Homogeneous ensembles consist of multiple instances of the same base learning algorithm, but with different parameters or subsets of data, whereas heterogeneous ensembles combine different types of models.
entity linking	task of predicting missing or potential links between entities in a network or graph. It is commonly applied in social network analysis, recommendation systems, and information retrieval. The goal is to infer or predict connections or relationships that are likely to exist but are not explicitly stated in the available data.
entity representation	representation of the semantic meaning, context, and relationships associated with entities. This can involve encoding entities as numerical vectors or embedding them into a high-dimensional space. Various approaches can be used, such as pre-trained word embeddings, entity embeddings, or contextualized embeddings generated by models like BERT or ELMO.
entity sense disambiguation (ESD)	task of determining the correct sense or meaning of an ambiguous entity mention in a given context
feature selection	process of selecting a subset of relevant and informative features from the raw text data to use in building machine learning models. The goal is to reduce the dimensionality of the data while retaining the most important information, which can lead to improved model performance, faster training times, and enhanced interpretability. In NLP, the features typically represent words, n-grams (groups of adjacent words), or other linguistic patterns that are extracted from the text data.
few-shot classification	task of classifying data into classes that were not seen during training, (zero-shot) or only a limited number of labeled examples per class (few-shot). This is achieved by leveraging additional information such as class descriptions, attributes, or semantic embeddings to generalize to unseen classes.
fine-grained entity recognition	identifying and categorizing specific, specialized entities within a text. Unlike general named entity recognition (NER), which focuses on identifying broad categories like person names, organizations, and locations, fine-grained entity recognition aims to identify more specific types of entities within these categories.
graph representation	structuring data as a set of nodes and edges
graph-based ranking	assigns importance or relevance scores to nodes in a graph. It involves representing the information or data as a graph, where nodes represent entities (e.g., documents, web pages, or words) and edges represent relationships or connections between the entities.In graph-based ranking, the importance of a node is determined by considering its connections to other nodes in the graph
hierarchical clustering	algorithm used to group similar data points into nested clusters or a hierarchical structure. It builds a tree-like structure known as a dendrogram, which represents the relationships and distances between data points.
hierarchical representation	representation of data in a way that reflects hierarchical relationships between different units of text, entities, concepts
hyperparameter optimisation	the process of finding the best values for the hyperparameters, external settings that define the behavior and configuration of a machine learning model to improve its performance and generalization on new data. This is achieved through techniques like grid search, random search, or Bayesian optimization.
keyphrase extraction (KE)	task of identifying and extracting key terms or phrases from a body of text. These keyphrases serve to summarize the content and topic of the text, and they can be used for a variety of applications, such as information retrieval, text summarization, and document clustering
language model	type of artificial intelligence (AI) model that is used to generate likely sequences of words, to score sequences of words, and to embed words and sentences in continuous vector spaces
latent variable	underlying, unobservable variable that is not directly measured or observed but is inferred from the available data. Latent variables are used to represent concepts, factors, or attributes that cannot be directly measured but play a crucial role in explaining the observed relationships among variables. Instead of directly observing the latent variable, researchers or models estimate its value based on the observed data and assumptions about the relationships among variables.
lemmatization	reducing words to their base or dictionary form, known as the lemma. The lemma represents the canonical or morphological root of a word, and lemmatization aims to normalize different inflected forms of a word to its base form.
multi-label classification	type of classification problem where each instance in the dataset can be associated with multiple labels or classes simultaneously. In contrast to traditional single-label classification, where each instance is assigned to only one class, multi-label classification allows for more complex and flexible predictions.
N-gram	refers to a contiguous sequence of n items, where an item can be a word, character, or any other unit of text.Used to capture the local or contextual information within a text by considering sequences of words or characters.
named entity recognition (NER)	task of identifying and classifying named entities in text into predefined categories such as person names, organizations, locations
negative sampling	used to address the problem of imbalanced datasets where the number of negative examples significantly outweighs the number of positive examples (e.g. in binary classification, word embeddings)
node centrality score	measure used in network analysis to quantify the importance or centrality of a node (vertex) in a graph. It aims to identify nodes that play critical roles in the overall structure and functioning of the network.
node embeddings	vectorized representations of nodes in a graph
part-of-speech (POS)	task of labeling the words in a text according to their grammatical role or part of speech. This includes categories such as noun, verb, adjective, adverb, pronoun, preposition, conjunction, interjection. POS tagging is often a crucial preliminary step in NLP pipelines, as it can provide valuable syntactic information that can improve performance on tasks such as Named Entity Recognition (NER), parsing, machine translation, and sentiment analysis.
phrase representation	representation typically in the form of vectors (lists of numbers) that capture the semantic meaning of the phrase
position embeddings	vector representations that encode the positional information of elements in a sequence. They are widely used in NLP models to capture the ordering or relative positions of words or tokens within a sentence or document. Position embeddings enable the models to consider the sequential nature of text data and capture dependencies between words based on their positions.
ranking	assigning a rank or ordering to a set of items, usually based on some criteria or measure of importance or relevance
regularization	technique used in machine learning to prevent overfitting and improve the generalization performance of a model. Overfitting occurs when a model learns the training data too well and fails to generalize well to unseen data.
relation extraction (RE)	task of identifying and categorizing semantic relationships between entities in a text. These entities are usually identified beforehand by a Named Entity Recognition (NER) system.
scoring	assigning a numerical value or "score" to a piece of data (like a word, sentence, or document) based on some criteria. This score is typically used to rank or order data, make decisions, or evaluate performance
seed words	initial example words used to bootstrap various NLP tasks such as text classification, named entity recognition, sentiment analysis, topic modeling, and more. They guide model learning by providing representative words for specific categories, sentiments, or topics.
sentence representation	representation typically in the form of vectors (lists of numbers) that capture the semantic meaning of the sentence
sentiment analysis	technique used to determine the sentiment or subjective information expressed in a piece of text. It involves automatically analyzing and classifying text data to identify the overall sentiment as positive, negative, or neutral.
sentiment lexicon	resource that associates words or phrases with sentiment information, indicating the positive, negative, or neutral sentiment of terms. It serves as a lookup table for sentiment analysis tasks to determine the sentiment orientation of text data.
similarity	degree to which two pieces of text are alike in terms of their meaning or semantic content
stemming	process of removing prefixes, suffixes, or inflections from words to obtain a common root form, often resulting in truncated but non-linguistically meaningful words, in contrast to lemmatization which produces valid and meaningful base forms.

stop words	commonly used words in a language that are considered to be uninformative or irrelevant in the context of text analysis and NLP
syntactic dependency parsing	technique that analyzes the grammatical structure of a sentence by identifying the syntactic relationships between words. It aims to determine how words in a sentence are connected to each other and how they depend on one another to form meaningful phrases or clauses. The output of syntactic dependency parsing is a dependency tree or graph that represents the syntactic structure of the sentence.
tagging	associating pieces of text, such as words, sentences, or documents, with informative labels, or "tags". These tags can represent a wide range of information, depending on the task at hand.
taxonomy	hierarchical classification of entities, concepts, or things, which are organized in parent-child relationships (typically a tree structure), where each parent can have multiple children, but each child has only one parent
term position weight	concept used in information retrieval and text analysis to assign higher weights or importance to terms based on their positions or proximity within a document. It aims to capture the notion that terms occurring closer to each other are likely to be more relevant or have stronger semantic relationships.
term-document matrix	illustrates the frequency of terms that occur in a collection of documents. In a term-document matrix, rows correspond to terms (words or phrases) in the corpus, while columns correspond to documents
text embeddings	type of representation used in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers. These vectors capture the semantic meaning of the words or phrases in a dense, low-dimensional space (typically a few hundred dimensions), as opposed to sparse, high-dimensional spaces (like one-hot encoding or Bag-of-Words representations)
topic modeling	task of discovering latent themes or topics in a collection of text documents. It allows us to extract high-level, abstract representations of the main subjects or ideas present in the text data.
transfer learning	machine learning technique where knowledge learned from one task or domain is leveraged to improve the performance on a different but related task or domain. The idea behind transfer learning is to transfer knowledge from a source task with abundant labeled data to a target task that has limited labeled data or is a different but related problem.
transformer	neural network architecture based on the self-attention mechanism, which allows it to capture relationships between different words in a sequence.Unlike recurrent neural networks (RNNs) that process sequential data sequentially, the transformer processes the entire sequence in parallel, making it highly efficientv
vector representation	numerical representation of input (image/ text/ etc) so that it can be processed by machine learning algorithms.
web crawler	automated software program used to browse and gather information from websites on the internet. It systematically visits web pages, follows hyperlinks, and extracts data for various purposes, such as indexing web content for search engines, data mining, or v
word co-occurence	frequency or occurrence of words appearing together in a given text or corpus. It is a measure of how often two or more words co-occur in close proximity within a specified context window

C Evaluation Measures

Name	Definition
Ablation Analysis	(also known as sensitivity analysis or feature importance analysis) technique used to understand the contribution or importance of individual components or factors in a model or system. It involves systematically removing or modifying specific components and observing the resulting impact on the overall performance or behavior.
Accuracy	metric used to evaluate the performance of classification models. It measures the proportion of correct predictions made by the model out of the total number of predictions. Accuracy is calculated by dividing the number of correctly classified instances by the total number of instances: Accuracy = (Number of Correct Predictions / Total Number of Predictions) * 100
Analysis of Variance (ANOVA)	Statistical technique used to analyze and compare the means of two or more groups or populations. It determines whether there are statistically significant differences among the group means based on the variance observed in the data.
Area Under Curve (AUC)	metric to evaluate the performance of binary classification models, particularly in situations where the class distribution is imbalanced. It measures the overall quality of the model's predictions across different discrimination thresholds. The AUC represents the area under the Receiver Operating Characteristic (ROC) curve. The ROC curve is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds.
Bilingual Evaluation Understudy (BLEU)	metric used to evaluate the quality of machine-generated translations by comparing them to reference translations. It measures the degree of overlap between the generated translation and the reference translations in terms of n-grams (contiguous sequences of words). It has been adapted and used in other text generation tasks as well. It can potentially be used to compare two pieces of text in situations where a reference or ground truth is available for evaluation.
Chi-Square	statistical measure used to assess the association between a candidate keyphrase and the documents in which it appears. It helps identify which terms or phrases are more likely to be important and informative within a specific document collection.
Clustering Accuracy (Purity)	measure used to evaluate the performance of clustering algorithms. It assesses how well a clustering solution assigns data points to their correct clusters based on known ground truth or reference labels. Purity ranges from 0 to 1, where higher values indicate better clustering performance. limitations: assumes that the number of clusters is equal to the number of true classes
Expansion Ratio	metric evaluates how effectively an aspect extraction system expands from the initial seed terms to accurately identify related terms that belong to the same aspect. Measures the ratio of correctly extracted terms over seed terms. Expansion Ratio= (Number of Correctly Extracted Terms / Number of Seed Terms) ×100
F1 Score	metric that provides a single measure that balances both precision and recall in situations where both are equally important. It is especially useful in situations where the data set is imbalanced, i.e., the number of 'positive' samples is very different than the number of 'negative' samples. F1 = 2 * (precision * recall) / (precision + recall)
FPR (False Positive Rate)	metric used to evaluate the performance of a binary classification model. It measures the proportion of falsely predicted positive instances among all the instances that are actually negative. FPR = False Positives / (True Negatives + False Positives)
Harvest Rate (HR)	ratio of the number of retrieved relevant webpages over the total number of retrieved webpages, which measures how well the crawler doing at rejecting irrelevant webpages
Hit Ratio (HT)	evaluation metric commonly used in recommendation systems to measure the effectiveness of the recommendation algorithm in predicting or suggesting relevant items to users. It assesses whether the ground truth or desired item is included in the top-k recommendations provided to the user.
Jaccard Index	metric used to evaluate the overlap between predicted and true labels in multi-label classification, indicating how well the predicted labels match the actual labels for a given instance. Particularly useful when dealing with imbalanced classes and when you want to measure the overlap between predicted labels and true labels. For a single example, the Jaccard index is calculated as the size of the intersection of the predicted labels and the true labels divided by the size of their union.
Logarithmic Loss	commonly used metric for evaluating the performance of probabilistic classification models. It measures the accuracy of a model's predicted probabilities by comparing them to the true class labels.
Label Ranking Average Precision (LRAP)	metric used to evaluate the performance of multi-label classification tasks. It measures the precision of correctly ranked labels for each instance, taking into account the order of the predicted labels.
Mean Absolute Error (MAE)	metric used to evaluate the performance of a regression model. It measures the average absolute difference between the predicted values and the actual values in a dataset. The lower the MAE, the better the model's predictions.
Mean Average Precision (MAP)	metric commonly used to evaluate the performance of information retrieval systems, such as search engines or recommender systems. It assesses the quality of the ranked results by considering both precision and recall. MAP calculates the average precision (AP) for each query or user, and then takes the mean of these average precision values across all queries or users. It measures how well the system retrieves relevant items and ranks them higher.
Mean Reciprocal Rank (MRR)	evaluation metric commonly used in information retrieval and ranking tasks. It measures the effectiveness of a system in ranking items based on their relevance to a given query.
Mean Squared Error (MSE)	metric used to measure the average squared difference between predicted values and the actual values in a regression task. It provides a measure of how well a regression model fits the observed data by quantifying the average magnitude of the prediction errors. The MSE is calculated by taking the average of the squared differences between the predicted values (y_pred) and the actual values (y_true) for a set of data points: MSE = (1/n) * Σ(y_pred - y_true)²
Normalised Discounted Cumulative Gain (NDCG)	metric commonly used to evaluate the effectiveness or ranking quality of search engines, recommendation systems, or information retrieval systems. It measures the quality of the ranked list of items or documents based on their relevance to a given query or user.
Normalized Mutual Information (NMI)	evaluation metric used to assess the quality of a clustering solution by comparing it to a set of ground truth or reference labels. NMI measures the mutual information between the clustering assignments and the true labels, normalized to take into account the sizes of the clusters and the true classes.
Partial Matching	evaluation metric used to assess the similarity or overlap between two sequences or sets. It allows for partial credit or partial matching of elements, taking into account partial similarities or partial matches instead of requiring an exact match. Examples of partial matching metrics include the Jaccard similarity, Sørensen-Dice coefficient, cosine similarity, and the Fuzzy matching algorithms like Levenshtein distance or Jaro-Winkler distance.
Perplexity	measure commonly used to evaluate the performance of language models, including topic models. It quantifies how well a language model predicts a given sequence of words or documents. Lower perplexity values indicate better performance, meaning the language model is more accurate and has a better understanding of the data. = exponential average negative log probability of the corpus under the model
Precision	(also known as positive predictive value) metric used in statistics and machine learning to measure the effectiveness of a classification model. In the context of a binary classification problem (where instances are classified as either positive or negative), precision is defined as the proportion of predicted positive instances that are actually positive. Precision = True Positives / (True Positives + False Positives)
Recall	(also known as sensitivity, hit rate, or true positive rate (TPR)) metric used in statistics and machine learning to measure the effectiveness of a classification model. Recall = True Positives / (True Positives + False Negatives)
Recall-Oriented Understudy for Gisting Evaluation (Rouge-L)	metric used to evaluate the quality of automatic text summaries or machine-generated text by comparing them to reference summaries. It measures the similarity between the generated summary and one or more reference summaries by considering the longest common subsequence (LCS) between them.
Receiver Operating Characteristic (ROC)	graphical representation used to evaluate the performance of binary classification models or classifiers. It plots the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds. A classifier with a higher AUC (area under curve) generally indicates better performance in distinguishing between positive and negative instances.
Residual MEan Square Error (RMSE)	metric to evaluate the accuracy of regression models. It measures the average magnitude of the differences between the predicted values and the actual values (residuals) in a regression task. RMSE is calculated by taking the square root of the mean of the squared differences between the predicted values (y_pred) and the actual values (y_true) for a set of data points: RMSE = sqrt((1/n) * Σ(y_pred - y_true)²)
Shortest Path Kernel score (SP Kernel)	similarity measure used in taxonomy comparison or hierarchical structure analysis. It quantifies the similarity between two taxonomies or hierarchical structures based on the concept of shortest paths.
Specificity	(also known as true negative rate) metric used to evaluate the performance of a binary classification model, particularly in situations where the negative class is of interest. It measures the proportion of correctly classified negative instances (true negatives) out of all the actual negative instances. Specificity = True Negatives / (True Negatives + False Positives)
Support	metric that quantifies the frequency or occurrence of an itemset or rule in a given dataset. It provides an indication of how often the class/ label appears in the dataset relative to the total number of data points.
t-test	statistical test used to determine whether there is a significant difference between the means of two groups or samples. It is commonly used when comparing the means of two independent groups or when comparing the mean of a single group to a known population mean.
Weisfeiler-Lehman Kernel score (WL Kernel)	similarity measure used in graph comparison and graph classification tasks. It quantifies the similarity between two graphs by considering their structural properties and neighborhood information.

D Tools/ Libraries/ Resources

Name	Definition
CoreNLP Tool	a software suite provided by Stanford that provides a set of natural language analysis tools. It can perform tasks such as part-of-speech (POS) tagging, named entity recognition (NER), parsing (both dependency and constituency parsing), co-reference resolution, and sentiment analysis, among others. It's designed to be highly flexible and extensible, with the ability to add custom processing modules or annotation types. It can be used via an API in a Java program, or run as a standalone web service.
WordNet	a lexical database of English words, which groups words into sets of synonyms called synsets, provides short definitions, and records the various semantic relationships between these synonym sets. The goal of WordNet is to build a useful representation of semantic memory, similar to how humans remember and use words and their meanings.
NLTK	an open-source library in Python that provides tools and resources for working with human language data. NLTK offers a wide range of functionalities for tasks such as text preprocessing, tokenization, stemming, lemmatization, part-of-speech tagging, syntactic parsing, named entity recognition, sentiment analysis, machine translation, and more.
SpaCy	an open-source software library for advanced NLP, written in Python. It's designed to help you perform a variety of NLP tasks such as part-of-speech tagging, named entity recognition, and dependency parsing, among others.
Gensim	an open-source python library specifically designed for topic modeling, document similarity analysis, and NLP tasks. Gensim provides support for word embeddings, which are vector representations of words that capture semantic and syntactic relationships. It allows you to train your own word embeddings using algorithms like Word2Vec and FastText or load pre-trained word embeddings such as Word2Vec, GloVe, or FastText models.
fastText	an open-source library that provides efficient tools for word embedding and text classification. It uses a skip-gram model with character n-grams to learn continuous representations for words, even for rare or out-of-vocabulary words. It employs a bag-of-words approach and learns a classifier using the word embeddings as input features. The classification model can handle multi-class and multi-label classification, and it supports hierarchical classification as well.
Sci-kit Learn	an open-source python library for machine learning. It provides a comprehensive collection of tools and functionalities for various stages of the machine learning workflow, including data preprocessing, feature selection, model training, evaluation, and deployment. Designed to work well with other Python libraries for data analysis and scientific computing, such as NumPy and pandas. It also provides interoperability with libraries like TensorFlow and PyTorch for deep learning.
DBpedia	a community-driven project that extracts structured data from Wikipedia and makes it available as Linked Data. It aims to transform the unstructured information in Wikipedia into a structured and machine-readable format, which can be easily queried and linked to other datasets on the web.
ProgrammableWeb	a popular online directory and platform that provides information, resources, and services related to web APIs (Application Programming Interfaces). It serves as a comprehensive source for developers, businesses, and enthusiasts seeking APIs for building web and mobile applications, integrating services, and accessing data from various platforms. *On February 3, 2023 Mulesoft announced that after 17 years in operation, it had shut down Programmable Web
Freebase	a collaborative knowledge graph and database.structured and comprehensive collection of information about entities, such as people, places, organizations, and more. Freebase allowed users to contribute and edit information, and it served as a source of data for various applications and services. In 2014, Freebase was deprecated and replaced by Wikidata
Wikidata	a collaborative and multilingual knowledge base that serves as a central repository of structured data for Wikimedia projects
SentiWordNet	a lexical resource for sentiment analysis and opinion mining in natural language processing. Serving as an extension of WordNet, it assigns sentiment scores to words in WordNet based on their positivity, negativity, and neutrality. Each word in SentiWordNet is associated with a set of three scores, representing its positive, negative, and objective (neutral) polarities. These scores indicate the degree of sentiment expressed by the word.
Bing Liu Opinion Lexicon	a lexicon or dictionary that contains a collection of words or phrases along with their associated sentiment polarity.
Stanford OpenIE	a module that performs Open Information Extraction, extracting relations from text using a set of pre-defined patterns and linguistic analysis. The OpenIE module in Stanford CoreNLP is designed to extract relations and generate triples in the form of (subject, relation, object) from sentences or documents.
SenticNet	a knowledge base that provides more detailed and multi-faceted sentiment information at the concept level, not just at the word level. For each concept, SenticNet provides a polarity value, includes semantic information about related concepts, and represents concepts as groups of related words, providing a more context-rich representation of sentiment.
BabelNet	al arge-scale, multilingual encyclopedic dictionary and semantic network where synsets (main meaning units) are connected via semantic relations.
Probase Knowledge Base	a large-scale probabilistic knowledge base that contains information about concepts, their attributes, and their relationships. It is designed to capture general knowledge in a structured manner and serves as a valuable resource for various NLP and machine learning tasks.

E Prompt Descriptions

E.1 Feature Classification Prompts

Prompt 1

Table 9: Prompt 1: Feature Classification

```
Considering cloud service selection and the few-shot examples below,
classify Term, as seen in Context:  Relevant (True/False)?
example:  CRM
Context:  Choose a CRM implementation partner that continues to grow
and adapt with you, helping you stand out from the rest.  / This CRM
was designed for small business.
Relevant:  True
example:  audit logs
Context:  Likewise, you can store audit logs for accessing the
Kubernetes API in an S3 bucket you define.  / With Cloud Audit Logs
integration, every interaction with Secret Manager generates an audit
log.
Relevant:  True
example:  resources
Context:  Distribute traffic across resources / All resources are
priced on an hourly basis, so you'll only pay for what you need, when
you need it.
Relevant:  False
example:  scratch
Context:  Create custom solutions without starting from scratch.  /
No need to start from scratch.
Relevant:  False
Term:  {term}
Context:  {" / ".join(contexts)}
Relevant?  Answer with one word
```

Prompt 2

Table 10: Prompt 2: Feature Classification

```
Considering cloud service selection and the NIST Cloud Reference
Architecture, classify Term, as a **standalone term**, given Context.
Is it a **cloud-specific** service feature, category, supported
software, certification or process (True/False)?
Examples:
* True :  CRM (Customer Relationship Management) - Software
specifically designed for managing customer data in the cloud.
* True :  Audit Logs - Feature for tracking activity within a cloud
service.
* True :  backup as a service - cloud service.
* True :  Linux - supported operating system.
* False :  algorithms - Finite sequence of rigorous instructions, too
general.
* False :  public internet access - Related, but not cloud-specific.
* False :  queue - Ambiguous as standalone.
* False :  execution - Ambiguous as standalone.
Term:  {term}
Context:  {" / ".join(contexts)}
Classification?  Answer with one word (True/False)
```

Prompt 3

Table 11: Prompt 3: Feature Classification

```
Considering cloud service vendor selection and the NIST Cloud
Reference Architecture, classify Term, as a **standalone term**,
given Context.  Is it a cloud feature, category, supported
software or language, functionality, certification or process of a
vendor, that a potential customer would be interested in knowing
(True/False)?  If the term is not meaningful/ query-able by itself,
return False.  The provided context is not always representative, do
not base the answer on it (avoid over-classification)
Examples:
* True :  CRM (Customer Relationship Management) - Software
specifically designed for managing customer data in the cloud.
* True :  Audit Logs - Feature for tracking activity within a cloud
service.
* True :  backup as a service - cloud service.
* True :  Linux - supported operating system.
* False :  algorithms - Finite sequence of rigorous instructions, too
general.
* False :  public internet access - Related, but not cloud-specific.
* False :  execution - Ambiguous as standalone.
* False :  cloud / cloud service / application / customer - too
general.  Even though they appear in the Reference Architecture,
we are interested in narrower categories.
* True:  Security / False:  cyberattack (not a vendor feature)
* False:  Intelligence / True:  Business Intelligence
* False:  GPU / True:  Parallel Processing
* False:  Performance / True:  Performance Metric or peak performance
etc.
Term:  {term}
Context:  {" / ".join(contexts)}
Classification?  Answer with one word (True/False)
```

Prompt 4

Table 12: Prompt 4: Feature Classification

```
Considering cloud service vendor selection and the NIST Cloud
Reference Architecture, classify Term, as a **standalone term**,
given Context.  Is it a cloud feature, category, supported
software or language, functionality, certification or process of a
vendor, that a potential customer would be interested in knowing
(True/False)?  If the term is not query-able and you would not
include it in a Knowledge Graph, return False.  The provided context
is not always representative, do not base the answer on it (avoid
over-classification).
Examples:
* True :  CRM (Customer Relationship Management) - Software
specifically designed for managing customer data in the cloud.
* True :  Audit Logs - Feature for tracking activity within a cloud
service.
* True :  Linux - supported operating system.
* False :  public internet access - not a cloud provider feature
* False :  execution / algorithm / CPU / Bytes / India :  not
meaningful as standalone
* True :  Parallel execution / Computing resources / Storage Capacity
/ Region Coverage
* False :  Cyber attack / Intelligence
* True:  Cyber Security / Business Intelligence
* False :  cloud / cloud service / application / customer :  in the
Reference Architecture, but we are interested in specific features
Term:  {term}
Context:  {" / ".join(contexts)}
Classification?  Answer with True or False
```

Prompt 5

Table 13: Prompt 5: Feature Classification

```
Considering cloud service vendor selection, classify Term, as a
**standalone term**, given Context.  Is it a cloud service feature,
category, supported software or programming language, certification
or other notable characteristic of a vendor (True/False)?.  The term
should be domain-specific and meaningful by itself.  The provided
context is not exhaustive, do not base the answer on it.  Avoid
overclassification and focus on technical terms.
Examples:
* True :  CRM (Customer Relationship Management) - Software
specifically designed for managing customer data in the cloud.
* True :  Audit Logs - Feature for tracking activity within a cloud
service.
* True :  Linux - supported operating system.
* False :  public internet access - not a cloud provider feature
* False :  execution / algorithm / CPU / Bytes / India :  not
meaningful as standalone
* True :  parallel execution / machine learning algorithm / Computing
resources / Storage Capacity / Region Coverage
* True:  Cyber Security / Business Intelligence
* False :  cloud / cloud service / cloud application / cloud customer
:  too general, looking for specific features
Term:  {term}
Context:  {" / ".join(contexts)}
Classification?  Answer with True or False
```

Prompt 6

Table 14: Prompt 6: Feature Classification

```
Considering cloud service vendor selection, classify Term, as a
**standalone term**, given Context.  Is it a cloud service feature,
category, supported software or programming language, certification
or other notable characteristic of a vendor (True/False)?.  The term
should be domain-specific, technical, meaningful by itself, convey
cloud-specific implementations and not common web service practices.
The provided context is not exhaustive, do not base the answer on it.
Examples:
* True :  CRM (Customer Relationship Management) - Software
specifically designed for managing customer data in the cloud.
* True :  Audit Logs - Feature for tracking activity within a cloud
service.
* True :  Linux - supported operating system.
* False :  public internet access - not a cloud provider feature
* False :  execution / algorithm / CPU / Bytes / India :  not
meaningful as standalone
* True :  parallel execution / machine learning algorithm / Computing
resources / Storage Capacity
* True:  Cyber Security / Business Intelligence
* False :  cloud / cloud service / cloud application / cloud customer
:  too general, looking for specific features
* False :  HTTP / port / HTML / CSS/ template :  domain relevant, but
common practices
Term:  {term}
Context:  {" / ".join(contexts)}
Classification?  Answer single wordedly with True or False
```

E.2 Match Quality Classification Prompts

Prompt 1

Table 15: Prompt 1: Match Quality Classification

```
Given two terms and the context they appear in, classify if they refer to
the same thing/ concept.
Examples:
* Term 1:  Scratch (programming language)
 Context 1:  Scratch is a high-level block-based visual programming
language and website aimed primarily at children as an educational tool for
programming, with a target audience of ages 8 to 16.
 Term 2:  scratch
 Context 2:  Create custom solutions without starting from scratch.  / No
need to start from scratch.
 Answer:  False
* Term 1:  CRM software
 Context 1:  Customer relationship management (CRM) is a process in which a
business or other organization administers its interactions with customers,
typically using data analysis to study large amounts of information.
 Term 2:  CRM
 Context 2:  Choose a CRM implementation partner that continues to grow
and adapt with you, helping you stand out from the rest.  / This CRM was
designed for small business.
 Answer:  True
* Term 1:  Hypervisor
 Context 1:  A hypervisor (also known as a virtual machine monitor, VMM,
or virtualizer) is a type of computer software, firmware or hardware that
creates and runs virtual machines.
 Term 2:  hypervisor
 Context 2:  Regardless of which hypervisor you choose, we manage and
support your virtualized configuration, 24x7x365.  / Hypervisor :  KVM on
Linux
 Answer:  True
* Term 1:  Integrity
 Context 1:  Integrity is the practice of being honest and showing a
consistent and uncompromising adherence to strong moral and ethical
principles and values.
 Term 2:  integrity
 Context 2:  With business data scattered across the enterprise and value
chain, managing and maintaining its integrity can be difficult.  / Ensuring
your data and environments are properly segmented from other machines is
extremely important for the integrity of your data.
 Answer:  False (data integrity is not exactly the same with general
integirty described above)
Term 1:  {term1}
Context 1:  {abst_sent}
Term 2:  {term2}
Context:  {" / ".join(contexts)}
Answer?  provide single wordedly with True or False
```

Prompt 2

Table 16: Prompt 2: Match Quality Classification

```
Given two terms and the context they appear in, classify if they could
possibly refer to the same, or related thing/ concept/ entity/ process.
Examples:
* Term 1:  Scratch (programming language)
 Context 1:  Scratch is a high-level block-based visual programming
language and website aimed primarily at children as an educational tool for
programming, with a target audience of ages 8 to 16.
 Term 2:  scratch
 Context 2:  Create custom solutions without starting from scratch.  / No
need to start from scratch.
 Answer:  False
* Term 1:  CRM software
 Context 1:  Customer relationship management (CRM) is a process in which a
business or other organization administers its interactions with customers,
typically using data analysis to study large amounts of information.
 Term 2:  CRM
 Context 2:  Choose a CRM implementation partner that continues to grow
and adapt with you, helping you stand out from the rest.  / This CRM was
designed for small business.
 Answer:  True
* Term 1:  Hypervisor
 Context 1:  A hypervisor (also known as a virtual machine monitor, VMM,
or virtualizer) is a type of computer software, firmware or hardware that
creates and runs virtual machines.
 Term 2:  hypervisor
 Context 2:  Regardless of which hypervisor you choose, we manage and
support your virtualized configuration, 24x7x365.  / Hypervisor :  KVM on
Linux
 Answer:  True
* Term 1:  Integrity
 Context 1:  Integrity is the practice of being honest and showing a
consistent and uncompromising adherence to strong moral and ethical
principles and values.
 Term 2:  integrity
 Context 2:  With business data scattered across the enterprise and value
chain, managing and maintaining its integrity can be difficult.  / Ensuring
your data and environments are properly segmented from other machines is
extremely important for the integrity of your data.
 Answer:  False (data integrity is not exactly the same with general
integirty described above)
* Term 1:  Global Namespace
 Context 1:  A Global Namespace (GNS) is a heterogeneous, enterprise-wide
abstraction of all file information, open to dynamic customization based on
user-defined parameters.
 Term 2:  namespace
 Context 2:  Administer Role-Based Access Control (RBAC) and oversee
namespace management.  / Aggregated namespace bringing together multiple
file sources
 Answer:  True (even though the first term is more specific, the phrase
'aggregated namespace' suggests the same concept)
Term 1:  {term1}
Context 1:  {abst_sent}
Term 2:  {term2}
Context:  {" / ".join(contexts)}
Answer?  provide single wordedly with True or False
```

Prompt 3

Table 17: Prompt 3: Match Quality Classification

```
Given two terms and the context they appear in, classify if they refer to
the same, or directly related thing/ concept/ entity/ process.  Ideally, the
first term should provide context/ explanation for the second term, and not
be more specific.
Examples:
* Term 1:  Scratch (programming language)
 Context 1:  Scratch is a high-level block-based visual programming
language and website aimed primarily at children as an educational tool for
programming, with a target audience of ages 8 to 16.
 Term 2:  scratch
 Context 2:  Create custom solutions without starting from scratch.  / No
need to start from scratch.
 Answer:  False
* Term 1:  CRM software
 Context 1:  Customer relationship management (CRM) is a process in which a
business or other organization administers its interactions with customers,
typically using data analysis to study large amounts of information.
 Term 2:  CRM
 Context 2:  Choose a CRM implementation partner that continues to grow
and adapt with you, helping you stand out from the rest.  / This CRM was
designed for small business.
 Answer:  True
* Term 1:  Hypervisor
 Context 1:  A hypervisor (also known as a virtual machine monitor, VMM,
or virtualizer) is a type of computer software, firmware or hardware that
creates and runs virtual machines.
 Term 2:  hypervisor
 Context 2:  Regardless of which hypervisor you choose, we manage and
support your virtualized configuration, 24x7x365.  / Hypervisor :  KVM on
Linux
 Answer:  True
* Term 1:  Mac OS X Tiger
 Context 1:  Mac OS X Tiger (version 10.4) is the 5th major release of
macOS, Apple's desktop and server operating system for Mac computers.
 Term 2:  Mac OS
 Context 2:  Compatible with Windows, macOS and Linux / HiDrive software is
also available for MacOS.
 Answer:  False (first term is more specific than the second term)
* Term 1:  Global Namespace
 Context 1:  A Global Namespace (GNS) is a heterogeneous, enterprise-wide
abstraction of all file information, open to dynamic customization based on
user-defined parameters.
 Term 2:  namespace
 Context 2:  Administer Role-Based Access Control (RBAC) and oversee
namespace management.  / Aggregated namespace bringing together multiple
file sources
 Answer:  True (even though the first term is more specific, the phrase
'aggregated namespace' suggests the same concept)
Term 1:  {term1}
Context 1:  {abst_sent}
Term 2:  {term2}
Context:  {" / ".join(contexts)}
Answer?  provide single wordedly with True or False
```

F Cloud Service Feature Coverage

Cloud Services	Presence
Data management	✓
Analytics and BI	✓
Database	✓
Blockchain	✓
Data lakes	✓
Disaster recovery	✓
Infrastructure	✓
Network & content delivery	✓
Webhosting	✓
Virtual machines	✓
Virtual desktop	✓
Load balancers	✓
Domain name systems (DNS)	✓
Developer tools	✓
Commandline interface (CLI)	✓
Low-code applications	X
DevOps	✓
Business applications	✓
Mobile	✓
Mobile applications	✓
Web applications	✓
Internet of things (IoT)	✓
IoT analytics	X
IoT security	X
IoT device management	✓
IoT applications	X
Machine Learning and AI	X
Open AI	✓
Document/video/text reader	X
Chat bots	✓
Digital assistant	X
Running and deploying models	X
Management & Governance	✓
Cost management	✓
Monitoring	✓
Platform building	X
Media Services	X
Migration and transfer	✓
Data migration	✓
Application integration	✓
Security, Identity and Compliance	✓
Identity & Access management	✓
Security management	✓
Firewall management	✓
Serverless	✓

SLAs	Presence
Availability	✓
Access management	✓
PaaS ranking	✓
AI developer services	X
Analytics and business intelligence platforms	✓
Application performance monitoring and observability	✓
Cloud for ERP	✓
Data management systems	✓
Field service management	X
Procure-to-Pay Suites	X
Network firewalls	✓
Security information and event management	✓
Low-code application platforms	X
Cost management & optimization	✓
Public Cloud Container Platforms	X
API management	✓
Cloud development and infrastructure platforms	X
Identity & access management	✓
Public cloud IaaS	✓
Bring your own license (BYOL)	**Presence**
SQL Server	✓
Windows Server	✓
BizTalk	X
Microsoft Dynamics	X
SharePoint	✓
SPLA	X
Skype	X
Exchange	✓
System Center	X
Remote Desktop Services	✓
MSDN	X
Team Foundation Server	X
Project Server	X
Ubuntu	X
RHEL	X
RHEL SAP	X
SUSE	X
Oracle Database	X
IBM	X

Supported APIs	**Presence**
HTTPs API	✓
REST API	✓
Websocket API	✓
OpenAPI	✓
SOAP API	X
GraphQl API	X
gRPC	X
Bulk API	X
Pub/Sub API	X
ODATA	X
CORS	X
Apigee API	X
OAuth	X
Programming Language Support	**Presence**
Ruby	✓
C	X
C#	X
Swift	✓
Kotlin	X
PHP	✓
C++	X
node.js	✓
JavaScript	✓
Java	✓
smalltalk	X
PowerShell	✓
visual studio/WCF Abap	X
curl	X
Python	✓
TypeScript	X
Go	✓
Rust	X
iOS	✓
Haskell	X
perl	X
Scala	X
Clojure	X
postman	X
Jboss/xPaaS	X
.NET	✓
GraalVM	X

Databases (out of box)	**Presence**
mysql	✓
SQL Server	✓
IBM DB2	X
oracle	✓
Redis	✓
Postgres	✓
Apache Kafka	✓
nosql	✓
apache cassandra	✓
mongodb	✓
mariadb	✓
Compliance	**Presence**
GDPR	✓
CSA	✓
CIS benchmark C5	✓
GSMA SAS-SM	X
HECVAT	X
CyberGRX	X
CyberVadis	X
ISO 9001	✓
ISO 10012	X
ISO 14001	X
ISO 20000-1	X
ISO 22301	X
ISO 20243	X
ISO 27001	✓
ISO 27110	X
ISO 27701	X
ISO 27017	✓
ISO 27018	X
ISO 29151	X
ISO 31000	X
ISO 45001v ISO 50001	X
VPAT(WCAG)	X
SOC 1	✓
SOC 2	✓
SOC 3	✓
Service costs (Forecasting)	**Presence**
Pay per use	✓
Pay per hour	X
Pay per month	X
TCO calculator	✓
Cost optimisation	✓
Free trial	✓

Container orchestration	Presence
kubernetes	✓
Docker Swarm	✓
openshift	✓
cloud foundry	X
Apache Mesos	X
nomad	X
kontena	X
Operating Systems	**Presence**
Container-optimized OS	X
Solaris	X
IBM AIX	X
Hp-UX	X
Windows	✓
Linux	✓
Oracle linux	X
Red Hat Enterprise Linux (RHEL)	✓
openSUSE	X
SUSE Linux Enterprise Server	✓
Clearlinux	X
VMware esx	X
Arch linux	X
asianux	X
Fedora CoreOS	X
Fedora	✓
Ubuntu	✓
linux mint	X
Fortinet	X
anolis OS	X
FreeBSD	✓
CoreOS	X
Rocky linux	X
Alma linux	X
kali	X
CentOS	✓
centOS Stream	X
Bottlerocket	X
Debian	✓
MacOS	✓
Raspberry pi OS	X

Data Center Locations	Presence
North America	✓
United States	✓
Canada	✓
Mexico	X
South America	✓
Chile	X
Brazil	✓
Asia	✓
Russia	X
Israel	X
Qatar	X
Saudi Arabia	✓
Saudi Emirates	X
Taiwan	X
India	✓
Thailand	X
Singapore	✓
Indonesia	X
Philippines	X
South Korea	✓
Malaysia	X
Japan	✓
China	✓
Oceania	X
Australia	✓
New Zealand	X
Africa	✓
South Africa	X
Europe	✓
Netherlands	✓
Belgium	X
Denmark	X
Finland	✓
Germany	✓
France	✓
Spain	X
Greece	X
Italy	X
United Kingdom	✓
Poland	✓
Ireland	✓
Austria	X
Switzerland	✓
Norway	X
Sweden	X

G Bill of Materials (BoM)

This Bill of Materials outlines the resources and tools used in the development of this thesis.

Item No.	Component / Resource	Purpose / Notes
1	SLR document	Step-wise documenation of the literature review, availble at `https://doi.org/10.5281/zenodo.13899562`
2	GitHub repository	The code implementation, accessible at `https://github.com/ac56/auto_kg`