electronics

MDPI

Article

Processing the Narrative: Innovative Graph Models and Queries for Textual Content Knowledge Extraction †

Genoveva Vargas-Solar

CNRS, University of Lyon, INSA Lyon, UCBL, LIRIS, UMR5205, 69622 Villeurbanne, France; genoveva.vargas-solar@cnrs.fr

† This paper draws inspiration from a dialogue organized by the DOING and ROCED actions, supported by the CNRS GDR MADICS under the auspices of the French government. *An initial version of this paper, authored by G. Vargas-Solar, M. Halfeld Ferrari Alves, and A.L. Minard-Frost, was previously published on arXiv. This current version refines the discussion and concentrates on G. Vargas-Solar's perspectives on the topics addressed, gathering an overview of representative contributions in the different domains.*

Abstract: The internet contains vast amounts of text-based information across various domains, such as commercial documents, medical records, scientific research, engineering tests, and events affecting urban and natural environments. Extracting knowledge from these texts requires a deep understanding of natural language nuances and accurately representing content while preserving essential information. This process enables effective knowledge extraction, inference, and discovery. This paper proposes a critical study of state-of-the-art contributions exploring the complexities and emerging trends in representing, querying, and analysing content extracted from textual data. This study's hypothesis states that graph-based representations can be particularly effective when annotated with sophisticated querying and analytics techniques. This hypothesis is discussed through the lenses of contributions in linguistics, natural language processing, graph theory, databases, and artificial intelligence.

Keywords: text annotation; text processing; graph data models; graph analytics; data science queries on graphs

Citation: Vargas-Solar, G. Processing the Narrative: Innovative Graph Models and Queries for Textual Content Knowledge Extraction. *Electronics* **2024**, *13*, 3688. https://doi.org/10.3390/electronics13183688

Academic Editors: Arkaitz Zubiaga and Alberto Fernandez Hilario

Received: 14 May 2024
Revised: 20 August 2024
Accepted: 9 September 2024
Published: 17 September 2024

1. Introduction

Processing digital text involves more than just extracting its structural organisation; it encompasses representing, understanding, and elucidating the knowledge embedded within the narrative, by capturing language's structural and semantic nuances. The challenge lies in developing appropriate data structures and representations, including (i) numerical representations for quantifying "key" words, (ii) structural representations for depicting how words combine into sentences to construct meaning, and (iii) logical reasoning for understanding and validating discourse. Together, these elements enable narratives to be thoroughly explored and processed through algorithms, facilitating content analysis for a deeper comprehension and utilisation of textual content.

Text analysis encompasses processing textual data to extract machine-readable facts, thereby transforming unstructured, free-text content into structured data [1]. This involves decomposing heterogeneous documents into coherent, interpretable data elements. From a computational standpoint, text analysis leverages various methodologies, including semi-automatic language processing techniques, to address the inherent ambiguity in human language, knowledge representation strategies, and identifying patterns and trends that encapsulate knowledge within texts, collectively called querying. The objective of this study is to exhibit the pipeline that starts from content representation tasks towards the extraction of knowledge, its analysis and exploitation.

1.1. Contribution and Methodology

This paper proposes a multidisciplinary study of advances and complex challenges in representing, querying, and analysing textual data derived from diverse sources, including commercial, medical, scientific, and environmental texts available on the internet.

Successfully extracting and managing this knowledge demands a nuanced understanding of natural language alongside precise content representation that maintains integral information. The study emphasises the importance of synthesising approaches from multiple disciplines, such as linguistics, natural language processing, knowledge representation, data management, and analytics, to enhance the process of knowledge extraction from textual content, its exploration and discovery. It states the hypothesis that graph-based representations, once annotated, are particularly adapted to supporting advanced querying and analytics methods.

This paper examines the following aspects:

[i] The extraction and representation of text content using data models, including matrices with term frequencies, text graphs, ontologies, property graphs and vectorial representations through the design of graph databases.
[ii] The consequences of the chosen modelling approach on the capabilities of querying, updating, maintenance and knowledge discovery within the text content.

The paper articulates its state-of-the-art study through research questions that guide the analysis, discussions, and statements it presents.

RQ_1. What are the capabilities and limitations of graph-based representations (text graphs, ontologies, property graphs and vectorial representations) in capturing syntax and semantics within textual content? RQ_1 examines the barriers and challenges in processing textual content. It can be answered by exploring examples and identifying experiments that illustrate challenges in understanding these barriers.
RQ_2. How can a graph-based structural and semantic text representation address missing information, such as incomplete, low-quality, or contradictory annotations, and what are the implications for querying and maintenance? RQ_2 studies how to address missing information—such as incomplete, low-quality, or contradictory annotations—within text graphs, ontologies, property graphs or vectorial representations that structure text content. It considers the implications for querying and maintenance and identifies unresolved issues that demand further attention.
RQ_3. How do graph-based text representations enable the extraction of explicit and implicit knowledge? RQ_3 examines text content representations maintenance and update. It asks how annotations should evolve in response to changes in repositories and corpora and how newly deduced, discovered, or explicitly inserted knowledge can be validated.
RQ_4. What querying possibilities are enabled by structural and semantic text graph representations? RQ_4 explores the possibilities for querying different graph-based representations of text content. In the analytical sense, this includes querying graphs like traversals, pattern discovery, community discovery and centrality.

As said before, the paper explores these four research questions by integrating perspectives from NLP, Semantic Web, databases, and machine learning, identifying existing work, highlighting open issues, and demonstrating how these fields collectively transform textual content into actionable knowledge.

1.2. Organisation of the Paper

Accordingly, the paper is organised as follows. Section 2 elaborates on the problems, existing solutions and open issues on how to represent textual content by exploring its syntax and semantics. Section 3 addresses text content representation challenges from the semantics point of view. It discusses how the Semantic Web can contribute to representing knowledge from textual content and how it is maintained, queried and inferred. Section 4 discusses how graph database management systems can contribute building

graph databases to store text content and provide efficient methods to maintain, query and complete knowledge. Section 5 gives perspectives on how machine learning and artificial intelligence play a role in discovering knowledge within text content and from graphs. Section 6 concludes the paper and wraps up answers to the questions we used to discuss the text-to-knowledge transformation.

2. Structuring Textual Content into Graphs

Extracting knowledge from textual content refers to identifying, analysing, and transforming the information in written text into structured, meaningful insights or facts that can be used for decision-making, research, or further analysis. This process involves understanding the text at a deeper level, beyond just reading it, to derive actionable knowledge. For instance, in a collection of medical research articles, extracting knowledge might involve identifying and summarizing the key findings, such as the effectiveness of a treatment, the side effects observed, and the patient demographics most affected. This extracted knowledge can then inform clinical guidelines, design new studies, or provide insights into patient care.

Natural language processing (NLP) aims to enable computers to understand textual content, including the nuanced context within the language [2]. NLP techniques aim to accurately extract information and insights from text, facilitating the categorisation and organisation of associated documents. Language-processing systems [3] have historically been developed using symbolic methods, which involve hand-coding rules and dictionary lookup approaches, such as crafting grammatical rules or devising heuristic rules for stemming. Since the 90s, NLP research has relied on machine learning that calls for using statistical inference to learn linguistic rules through the analysis of large corpora. Corpora refers to the plural form of a corpus, which is a set of documents, possibly with human or computer annotations automatically.

The following sections explore extracting structured semantic information from text and converting it into text graphs, highlighting their application in querying. The section then addresses the specific challenges posed by language, such as nuances and ambiguity. It illustrates these concepts with an example working with medical cases that outlines the key tasks involved in structuring text. The section concludes with a discussion of current trends and open issues, answering the research questions that guide our study from a natural language processing perspective.

2.1. Extracting Structured Semantic Information

Structured semantic information extraction from text refers to the process of (semi) automatically identifying, extracting, and organizing meaningful information from unstructured text data into a structured format, often using predefined categories or entities. This process involves understanding the text's semantics—or meaning—rather than just its surface structure. Existing techniques process structured, semantic information through extraction techniques like NLP and machine learning to automatically identify relevant information (e.g., entities, concepts, relationships) and convert them into a structured format (e.g., a text graph).

2.1.1. Modelling Structural Text Content

Structured information refers to data organised in a clear, predefined format, such as tables, databases, or XML. For example, extracting details like names, dates, locations, and relationships between entities from a text and placing them into a structured format like a database. Two elements are essential for structuring textual content:

- the techniques adapted for processing the text seeking for entities considering the characteristics of the language (i.e., grammar) and;
- identifying the pertinent data structures for structuring the content.

[1] Processing structural textual content. NLP tasks have been developed for building structured information from text [4]. Techniques address the structural and morphological dimensions of texts and semantics representation. Regarding structure and morphology, NLP tasks include the following:

- Text and speech processing [5] encompasses the conversion of text from various formats (image, sound clip, text) into a textual representation, typically comprising words or sentences. The complexity of this task varies depending on the linguistic characteristics of the language in which the content is produced. Tasks include optical character recognition, speech recognition, speech segmentation, text-to-speech, and word segmentation.
- Morphological analysis [6] breaks down words into their constituent morphemes or lemmas to create normalised representations or to identify parts of speech (POS) within sentences (e.g., noun, verb, adjective). Tasks include lemmatisation, morphological segmentation, part-of-speech tagging, and stemming.
- Syntactic analysis [7,8] aims to understand the structural aspects of natural languages. It involves generating a grammar describing the syntax of a language, identifying sentence boundaries within text chunks, and constructing parse trees (grammatical analysis) that represent relationships (dependencies) between words. Dependency parsing establishes relationships with or without probabilities, while constituency parsing produces probabilistic context-free or stochastic grammars. Tasks include grammar induction, sentence breaking, and parsing.

[2] Data structures for structuring textual content. Different data structures form modelling textual content (e.g., matrices and text graphs). However, the study focuses on using graphs as data structures for structuring textual content that can be adapted to query and extracting knowledge profiting from the mathematical characteristics of graphs.

Existing literature [9,10] acknowledges that text graphs can represent the co-occurrence of words in documents, knowledge graphs, and sentences as graphs. In each type of graph, edges may carry different semantics. For instance, nodes represent concepts, and edges represent semantic relations. In contrast, in a co-occurrence graph, nodes represent words, and edges indicate absolute or relative frequencies between connected words or n-word window intervals.

2.1.2. Modelling Textual Semantics

Textual semantics modelling creates structured representations of meaning within text by capturing the relationships between words, phrases, and sentences. This approach is crucial for enabling computers to understand, interpret, and generate human language in a way that aligns with how humans comprehend it. Using models such as vectors, graphs, or other formal structures, textual semantics modelling captures the nuances of meaning in text, making it possible for machines to perform various language-based tasks. These tasks include information retrieval, question answering, text summarization, sentiment analysis, and machine translation, where preserving the original meaning across languages is essential. Three main tasks can be considered for modelling textual semantics:

1. Lexical semantics of individual words in context [11]: focuses on determining the computational meaning of individual words within a given context or from data. It involves identifying proper names (e.g., people or places) and their types (e.g., person, location, organisation), extracting subjective information (e.g., sentiments), identifying terminology, and disambiguating words with multiple meanings. This complex task includes lexical semantics, distributional semantics, named entity recognition, sentiment analysis, terminology extraction, word-sentence disambiguation, and entity linking.
2. Relational semantics [12]: involves identifying relationships among named entities, disambiguating semantic predicates and semantic roles, and generating formal rep-

resentations of text semantics. This task includes relationship extraction, semantic parsing, and semantic role labelling. For example, understanding that "bank" can mean a financial institution or the side of a river depending on the context, and modelling these different meanings accordingly.

3. Discourse analysis [13] extends beyond individual sentence semantics and includes tasks such as co-reference resolution, discourse parsing to determine discourse relationships between sentences, recognising textual entailment, identifying topics within text segments, and identifying argumentative structures. For example, in a sentence such as "He entered John's house through the front door", "the front door" is a referring expression. The bridging relationship to be identified is that the door referred to is the front door of John's house (rather than of some other structure that might also be referred to).

Textual content semantics modelling with graphs. Graph representations of textual content are powerful tools for modelling textual semantics because they can capture and represent the complex relationships between entities, concepts, and their contexts.

In a graph representation, nodes can represent entities, concepts, or key terms within the text. For instance, in a sentence like "The cat chased the mouse", the nodes might represent "cat", "chased", and "mouse". Each node encapsulates the semantic meaning of a particular word or phrase in the text. Edges in the graph represent the relationships between these entities or concepts. For example, in the same sentence, an edge might connect "cat" to "chased" and "chased" to "mouse", indicating the action and its participants. These relationships are key to understanding the text's semantics because they show how concepts are linked within the context.

Text graph representations can support inference by deriving new relationships or conclusions based on the graph's structure. For example, if we know that "Socrates is a man" and "All men are mortal", a graph structure can help infer that "Socrates is mortal". This capability is crucial for more advanced natural language understanding tasks where the system needs to derive new insights from existing information.

Graphs are inherently flexible and can be expanded or modified easily as new data becomes available. They are scalable, meaning they can represent not just small texts but also large bodies of knowledge, making them ideal for tasks like semantic search, where understanding the relationships between concepts across large documents is crucial.

2.2. Building a Text Graph

A text graph represents a text item, such as a document, passage, or sentence, as a graph. Building a text graph is a preprocessing step in NLP to support text condensation, term disambiguation, topic-based text summarization, relation extraction, and textual entailment. For creating a text graph, it is necessary first to identify the entities in the text that will be represented as nodes in the graph and then to determine the relationships between those entities that will be modelled as edges between the nodes. The specific types of entities and relationships used will depend on the context and task. Entities can take various forms, such as individual words, bigrams, n-grams, or sequences of variable lengths. Relationships can represent a variety of connections between entities, such as adjacency in a sentence, co-occurrence within a fixed-length window, or some semantic or syntactic relationship. A directed unweighted graph (see Figure 1) can be used for representing the following text (without stop words). This example was proposed in https://towardsdatascience.com/keyphrase-extraction-with-graph-centrality-989e142ce427, accessed on 19 August 2024:

> *Centrality indices answer the question, "What characterises an important vertex?" The answer is given in terms of a real-valued function on the vertices of a graph, where the values produced are expected to provide a ranking that identifies the most relevant nodes. The word "importance" has a vast number of meanings, leading to many different definitions of centrality.*

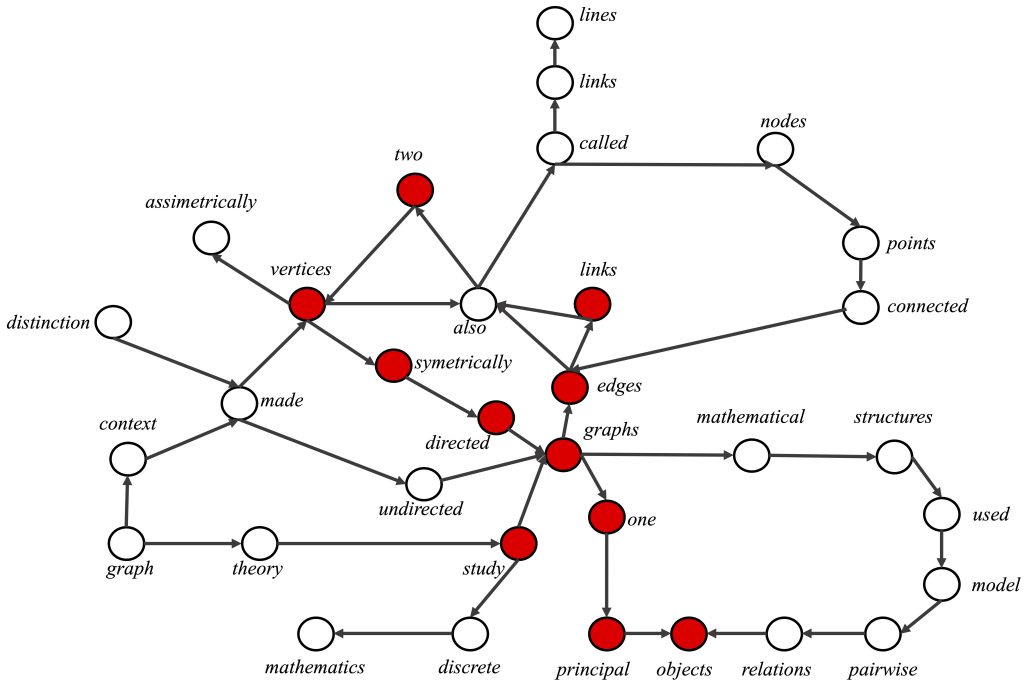

Figure 1. Text graph example.

By applying a centrality model, the nodes can be ranked based on their importance or relevance. The top-scoring nodes are then identified as potential keywords. The exact number of nodes considered as candidate keywords varies depending on the specific task. For example, in the figure, one-third of the total nodes (red nodes) are selected as candidate keywords based on their ranking.

Text graphs are used in a variety of applications to represent and analyse textual data:

- Information retrieval: represent the relationships between documents and the terms they contain, allowing for more effective information retrieval process.
- Text summarization: represent the relationships between sentences in a document, allowing for the automatic generation of summaries.
- Topic modelling: represent the relationships between documents and the topics they contain, automatically identifying topics within a corpus of text.
- Sentiment analysis: represent the relationships between words and their sentiment, allowing for automatic sentiment analysis in text.

2.3. Querying Text Graphs

Text content can be represented as different types of graphs, and every kind of graph has specific methods for querying and retrieving information.

2.3.1. Exploring Text Graphs

Regular path queries (RPQs) are a way to navigate over paths in a graph using regular expressions [14]. A regular path query (RPQ) is a regular expression q that returns all node pairs (u, v) from a graph database that is connected by an arbitrary path labelled with a word from L(q). Intuitively, in an RPQ, the input is a graph, a source node and a regular expression. The goal is to identify all nodes connected to the source through a simple path whose label sequence satisfies the given regular expression. The regular expression is a formal specification of the search space that interests the user. For example, one could use an RPQ to find all paths between two nodes that have a specific node in between or to find all nodes connected by paths of a specific form (https://iccl.inf.tu-dresden.de/w/images/7/7f/Lecture-17-rpqs.pdf, accessed on 19 August 2024).

In addition to RPQ, other types of queries can be used on text graphs:

- Attributed text graph queries extract information from attributed graphs, where vertices and edges are associated with attributes such as types, numbers, and texts. For example, suppose we have an attributed graph representing a document, where each node represents a word and has an attribute "word" containing the word itself. Edges represent the order of words in the document. We can use an RPQ to find all occurrences of a specific phrase in the document. If we want to find all occurrences of the words "artificial intelligence" in the document, we can use the following RPQ:

```
[word="artificial"]/./[word="intelligence"]
```

- Structure extraction is used to extract the underlying structure of a text graph by identifying and grouping equivalent nodes. For example, we can assume that we have a graph as described in the previous example. Suppose we want to group all occurrences of the word "artificial" together. In that case, we can use structure extraction to identify all nodes representing the word "artificial" and group them into a single node. This would result in a simplified graph where all occurrences of the word "artificial" are represented by a single node. Similarly, we can use structure extraction to group nodes representing equivalent concepts, such as synonyms or related words. We could group nodes representing the words "artificial intelligence", "AI", and "machine learning" into a single node representing the concept of artificial intelligence.

2.3.2. Analysing Text Graphs

By converting free text into a graph representation, the implicit structure of the text is made explicit. This representation allows immediate access to information such as the most commonly used words (degree), the most frequently used n-grams, and the words most commonly used to convey information (paths in the graph between every two nodes), among other things.

For example, extracting key phrases to return a list of relevant phrases (one or more words) from the input text can be possible. In this context, a relevant phrase is defined as one that is composed solely of candidate keywords, which are the most relevant unigrams. These candidate keywords are determined by graph centrality algorithms, which use the graph's structure to assign scores to nodes. In this case, the nodes represent words in the text and the relationships between them. Therefore, it is necessary to score the graph nodes (i.e., words) with a graph centrality algorithm [15,16], extract candidate keywords (i.e., most relevant individual words), extract key phrases (based on the candidate keywords).

In the previous example (see Figure 1), each node in the graph represents a term present in the input document (a single node for the multiple uses of the same term). Consider that after applying some centrality (centrality in a graph recognizes nodes that are important or central among the whole list of other nodes in a graph) algorithm, the top three nodes are identified: (1) graphs: 0.56, (2) study: 0.43, (3) objects: 0.35.

For extracting the candidate keywords, we can consider a third of the total number of nodes as the number of candidate keywords. Note that the choice depends on the context of the application. The candidate keywords (ordered from highest to lowest scoring) are shown in Figure 1:

```
1. graphs
2. study
3. objects
4. edges
5. vertices
6. two
7. link
8. directed
9. symmetrically
10. principal
11. one
```

A key phrase in the input document is considered relevant and can be used for various tasks, including extractive summarisation. It is defined as a sequence of words in the input document composed solely of candidate keywords. For instance, if both "mathematical" and "structures" are candidate keywords, then "mathematical structures" would be considered a key phrase, and the words would not be considered individually. In the previous text and considering the candidate keywords, examples of key phrases can be:

```
1. edges link two vertices symmetrically
2. directed graphs
3. edges link two vertices
4. principal objects
5. study
6. one
```

2.4. Processing the Nuances and Ambiguity of Natural Language

NLP methodologies still encounter several complex challenges that critically affect their ability to interpret textual content accurately. One of the primary obstacles is contextual ambiguity, where words and phrases may carry multiple meanings depending on their usage in specific contexts. This ambiguity complicates the task of determining the intended interpretation. Additionally, synonymy is challenging as different words with similar meanings can confuse text analysis.

Irony and sarcasm further complicate text interpretation for NLP systems, as these expressions require a nuanced understanding that algorithms often struggle to capture. General ambiguity within texts can also lead to misinterpretations, where NLP systems find it difficult to discern the correct meaning. Moreover, errors in text or speech, such as typos or grammatical inaccuracies, hinder the ability of NLP methods to process content effectively.

Another significant challenge arises from colloquialisms and slang, which vary widely across different regions and cultures and affect the accuracy of NLP interpretations. Similarly, domain-specific languages within various fields involve specialised terminologies that NLP systems must adapt to understand and process information accurately within those contexts.

This study examines the issue of bias in textual content and explores how text graphs can be utilized to address it. Additionally, it considers the challenge of managing the dynamic evolution of text content over time.

2.4.1. Dealing with Bias in NLP Processing

Alongside linguistic and technical challenges, there is the critical issue of bias within NLP systems [17]. Artificial intelligence models, particularly those that operate unsupervised, are prone to absorbing the inherent biases present in human language data, including racism, sexism, and ableism. This inadvertent capture of bias can perpetuate discrimination when such models are used in downstream NLP applications. Actively addressing these biases is crucial to ensuring fairness and equity in the outcomes of NLP methods, making it a fundamental aspect of advancing NLP technology. Although current research offers various methods for evaluating and mitigating biases across numerous tasks, there is still a significant need for a comprehensive understanding of these biases, the specific harms they cause, and how different evaluation methods compare [18]. NLP offers several techniques to mitigate bias, ensuring fairer and more equitable AI applications. Examples of these techniques are:

- identifying discriminatory language and biased patterns using sentiment analysis and entity recognition [19,20];
- preprocessing data to balance or neutralise biases, creating fairer datasets for training models [21];
- altering training procedures or model architectures ensures equitable performance across different demographic groups [22];
- using attention mechanisms can enhance the transparency of NLP models [23];

- helping to pinpoint and correct biases in how models process data [24];
- adjusting model training to calibrate outputs to reduce bias [25];
- establishing robust ethical frameworks ensures that NLP applications adhere to principles of fairness and equity [26].

Text graphs provide a structured and visual approach to uncovering biases in textual content. By mapping out relationships, analysing patterns, and quantifying associations, text graphs can expose both overt and subtle biases that may be embedded in language.

[1] Visualization of Relationships and Associations. Text graphs can map out the relationships between entities, concepts, and words in the text, making it easier to identify biased associations. For example, if certain adjectives (like "aggressive" or "emotional") are disproportionately associated with a particular gender or ethnicity, a text graph can visually highlight these associations, making the bias clear. By examining the structure of a text graph, one can spot imbalances in the representation of different groups or ideas. For instance, if the graph shows a significant clustering of negative sentiments around a specific demographic, it could indicate a bias in how that group is portrayed.

[2] Contextual Analysis. Text graphs can capture the context in which certain entities or groups are mentioned. Suppose certain stereotypes are consistently reinforced through specific relationships or descriptions (e.g., linking a profession predominantly to one gender). In that case, the graph will make these patterns visible, exposing potential biases in the text. By tracking the usage of specific terms across different contexts, text graphs can help identify biased language. For example, the graph can reveal if terms like "leader" are more frequently associated with one gender. In contrast, terms like "helper" are associated with one another, highlighting a bias in how roles are portrayed.

[3] Quantifying Bias through Metrics. By analysing the nodes (entities) and edges (relationships) in a text graph, one can quantify the extent of bias. Metrics like centrality, frequency, and clustering can indicate how often certain groups or ideas are connected to negative or positive terms, allowing for a more objective bias assessment. Text graphs can integrate sentiment analysis to evaluate how different entities are described. If the graph reveals that certain groups are frequently linked to negative sentiments or emotions, it can indicate underlying bias.

[4] Comparative Analysis. Text graphs allow for comparing biases across multiple documents or corpora. By comparing the structure and relationships in graphs from different sources, one can identify consistent biases or disparities in how certain topics or groups are represented. Text graphs can be used to track how biases evolve. By comparing graphs generated from texts written in different periods, one can see if biases have increased, decreased, or changed in nature, providing insights into how societal attitudes are reflected in language.

[5] Highlighting Ambiguity and Inconsistency. Text graphs can also highlight areas where the text is ambiguous or inconsistent in portraying specific ideas. For example, if a text graph shows conflicting relationships for a particular entity (e.g., an individual is described as both "competent" and "incompetent"), it could indicate a bias in how that entity is portrayed. Suppose the graph reveals that certain groups are represented inconsistently across different text parts (e.g., a minority group being portrayed positively in one section and negatively in another). In that case, this inconsistency might reflect an underlying bias in the content.

2.4.2. Maintaining Content Representation and Dealing with Annotations Evolution

Textual content is an evolving entity. Thus, maintaining and updating representations of text content and annotations involve complex, intertwined processes that demand human expertise and advanced computational techniques. This continuous challenge is critical in NLP, as it ensures the accuracy and currency of NLP methods. These maintenance tasks are vital for NLP applications' sustained effectiveness and reliability in accurately interpreting and processing language data.

In NLP, various representations of text content significantly boost performance and enhance accuracy across diverse applications. For example, vectorial representations such as word embedding to categorise documents into predefined groups in a multi-dimensional space efficiently facilitate precise querying of textual data [27] (see Section 5). These representations are useful for machine learning algorithms, which automatically discern relevant information from the text and suggest updates to improve content accuracy and coherence [28]. In NLP applications, such as text classification, these algorithms use representations like word embeddings to categorise documents into predefined groups efficiently [29]. Similarly, for information extraction tasks, NLP methods leverage these representations to identify and extract pertinent information from unstructured texts accurately [30,31], thus enhancing the overall effectiveness of the processing systems.

Text graphs offer a dynamic and visual way to represent the evolution of textual content. Text graphs can reveal how entities, relationships, and themes develop over time by incorporating temporal data, enabling real-time updates, and allowing for sequential or animated analysis. This capability is invaluable for understanding the progression of narratives, the evolution of discourse, or the changing focus within a body of text, providing deeper insights into how content evolves and what drives those changes.

Consider a scenario where a patient's medical history is documented over several years, including symptoms, diagnoses, treatments, and outcomes. By capturing the relationships between these different elements over time, a text graph can be constructed to model the evolution of the patient's disease.

By modelling the evolution of a patient's disease with a text graph, healthcare providers can gain a comprehensive and dynamic view of the patient's medical history. The graph not only captures the current state of the patient's condition but also allows for visualization of disease progression, analysis of treatment efficacy, and identification of patterns that can inform future care.

A text graph is constructed where:

- Nodes represent key entities, such as symptoms (e.g., "chest pain", "shortness of breath"), diagnoses (e.g., "coronary artery disease", "hypertension"), treatments (e.g., "beta blockers", "angioplasty"), lab results (e.g., "elevated cholesterol", "EKG results"), and visits (e.g., "initial consultation", "six-month follow-up").
- Edges represent relationships between these entities, such as "diagnosed_with", "treated_with", "led_to", and "associated_with". These edges can also carry temporal annotations to indicate when these relationships were recorded.

The graph starts with nodes representing the initial symptoms and the first diagnosis. For example, "chest pain" and "shortness of breath" are connected to a "diagnosed_with" edge leading to "coronary artery disease". The graph shows the treatments applied over time. For instance, "beta blockers" and "angioplasty" nodes are connected to the "treated_with" edge from the "coronary artery disease" node.

As the patient returns for follow-up visits, new nodes are added to the graph to represent changes in symptoms, the introduction of new diagnoses, and updates to the treatment plan. For example, after a follow-up visit, "new onset diabetes" might be added, linked to "prescribed metformin". Lab results and their impact on the diagnosis or treatment are also integrated into the graph. For example, an "elevated cholesterol" node might connect to both the "coronary artery disease" and "prescribed statins" nodes, showing how lab results influenced treatment decisions.

2.5. Wrap Up Example: Textual Content Representation, Maintenance and Exploration in Healthcare

Our study is driven by the goal of extracting meaningful insights from clinical case documents. To illustrate the challenges involved in transforming textual content into actionable knowledge for medical monitoring, we use a healthcare example to highlight key considerations in this process.

2.5.1. Processing Textual Content in Healthcare

NLP is extensively applied in healthcare, where it supports clinical documentation improvement, enhances clinical decision-making, and boosts patient safety measures (https://www.mckinsey.com/industries/healthcare/our-insights/natural-language-processing-in-healthcare, accessed on 19 August 2024):

[1] Clinical documentation: NLP significantly enhances the extraction and structuring of data from unstructured clinical notes, thereby improving patient care through streamlined analysis [32]. For instance, consider a patient's clinical notes that describe "severe chest pain" and "shortness of breath", mention the administration of "nitroglycerin", and recommend a "stress test". NLP technology can process this information by identifying and tagging medical terms for automatically detecting and classifying terms within the notes as symptoms (e.g., chest pain, shortness of breath), medications (e.g., nitroglycerin), and procedures (e.g., stress test). They can also structure the identified information into organised categories such as symptoms, diagnosed conditions, medications administered, and procedures recommended.

[2] Predictive analytics: NLP analyses electronic medical records to identify patients at heightened risk of health disparities, bolstering surveillance efforts.

[3] Clinical decision support: NLP aids clinical decision-making by furnishing clinicians with pertinent patient medical history information alongside the latest research findings and guidelines.

[4] Patient experience: NLP enhances the patient experience by delivering personalised information and support across the patient's care continuum.

There are several challenges associated the extraction of knowledge from clinical cases [33] including:

- Variation in language: Many different dialects and language variations are used in medical records, making it difficult for NLP algorithms to understand and interpret the content accurately.
- Data standardisation: Poor standardisation of data elements, insufficient data governance policies, and variation in the design and programming of electronic health records (EHRs) can make it challenging to use NLP to fill the gaps in structured data.
- Domain-specific language: The application of NLP methodologies for domain-specific languages, such as biomedical text, can be challenging due to the complexity and specificity of the language used.

2.5.2. Structural and Semantic Modelling of Clinical Cases

Structural and semantic modelling of clinical case texts using graphs provides a powerful solution to challenges such as language variation, data standardization, and the complexity of domain-specific language in medical records. Graphs can effectively map synonyms and contextually related terms, which is particularly useful in handling language variations across clinical documents. For instance, by linking terms like "MI" (myocardial infarction) and "heart attack", a graph ensures that the system recognizes these as the same condition despite differences in terminology.

Moreover, clinical case texts often originate from various electronic health record (EHR) systems, each with its format. Graphs can standardize and integrate these diverse data elements into a cohesive structure. By mapping patient data, lab results, diagnoses, and treatments to a common set of nodes and edges, graphs create a unified representation of clinical information, even when underlying EHR systems lack standardization. This harmonization allows for consistent data extraction and analysis, making it easier to draw meaningful insights across different records.

Graphs are adept at capturing the complex relationships and hierarchical structures inherent in domain-specific languages, such as those used in biomedical contexts. By representing these relationships within a graph, models can better understand and interpret the specialized language found in clinical case texts. For example, a graph might link

"antibiotics" to "penicillin" and further connect "penicillin" to specific treatment protocols, aiding in the accurate interpretation of medical terms.

2.5.3. Querying and Analysing Clinical Cases Represented as Graphs: From Facts to Knowledge

Applying graph-based models to clinical case texts provides healthcare professionals with more accurate and comprehensive insights, significantly supporting clinical decision-making. When a doctor enters a patient's symptoms and history into an EHR, the graph model can suggest possible diagnoses or treatment options by leveraging structured and semantically enriched data. This standardized data representation also helps identify patterns and trends that might indicate a patient's deteriorating condition, allowing for timely interventions. Additionally, researchers can analyse large datasets of clinical case texts using these models to uncover correlations, trends, and insights that might not be apparent through traditional methods. By examining the connections between treatments and patient outcomes, researchers can discover more effective treatment protocols, advancing medical research. The graph-based representation of clinical case texts also enables a wide range of traversal and analytics queries, facilitating deeper exploration of complex relationships within medical data. This capability leads to more informed decision-making and improved patient outcomes.

Text graph representations can generally answer seven types of traversal and analytics queries. Figure 2 shows use cases for every family of queries. Consider, for example, the case where a doctor wants to determine if a patient's recent symptoms and treatment pattern are likely to lead to a relapse of a condition like myocardial infarction (MI). Assume that there is a graph of clinical cases where nodes represent entities such as symptoms, diagnoses, treatments, and outcomes. Edges represent relationships such as "leads to", "treated with", or "results in". Consider the pattern matching and path query: Has there been a previous case where patients with the same symptoms and treatment experienced a relapse of MI within six months?

Query Type	Use case	Example
Path Queries	Identifying the most direct relationship between a reported symptom and the corresponding diagnosis in a patient's history.	"Find the shortest path between a symptom and a diagnosis."
Subgraph Queries	Analyzing the effectiveness of treatments for a particular condition by focusing on a specific subset of the data.	"Extract the subgraph that includes all treatments related to a specific diagnosis within the last year."
Pattern Matching Queries	Detecting patterns that may indicate ineffective treatments or the need for alternative therapies.	"Identify all patients who have a pattern of symptoms followed by a specific treatment and then a recurrence of symptoms."
Centrality & Ranking Queries	Identifying key symptoms most strongly associated with various conditions can be critical in early diagnosis.	"Which symptoms are most central to the network of related diagnoses?"
Aggregation Queries	Summarizing common treatment protocols and their outcomes for a particular diagnosis.	"What is the most common treatment path for patients diagnosed with a specific condition?"
Reachability Queries	Predicting potential critical outcomes based on a patient's historical data, allowing for preemptive measures.	"Can a patient's condition reach a critical outcome based on their current symptom and treatment history?"
Similarity Queries	Comparing a new patient's case with similar historical cases to predict outcomes and recommend treatments.	"Find patients with a similar pattern of symptoms and treatments to identify potential outcomes."

Figure 2. Text graph query types.

The query traverses the graph to match the pattern of symptoms and treatments in the current patient's history with those of other patients. It checks the outcomes within a specified time frame (e.g., six months). The query returns a list of similar cases where patients experienced a relapse of MI after a similar pattern of symptoms and treatment. The doctor can use this information to adjust the treatment plan to mitigate the risk of relapse.

2.6. Discussion: Current Trends and Open Issues

Efficiently representing syntax and semantics is crucial for NLP methods, enabling applications like question answering, automatic summarization, grammatical error correction, and transforming database information into readable text. Text mining and NLP techniques are essential for extracting structured semantic information from unstructured text, allowing the identification and analysis of concepts and relationships within the data. Standard methods include structure extraction, tokenization, lemmatization, and entity recognition. The power of NLP lies in machine learning algorithms, such as decision trees and neural networks, which underpin complex systems and enhance text analysis capabilities. As text data grows in volume and complexity, these techniques play a vital role in extracting meaningful insights and aiding in informed decision-making. Moreover, there is a growing focus in NLP on exploring the cognitive aspects of language, such as multilingualism and multimodality, and a shift towards advanced supervised and weakly supervised representation learning methods integrated into comprehensive end-to-end systems [34].

This study analyses the capabilities of graph-based representations for capturing syntax and semantics within textual content, enabling the extraction of both explicit and implicit knowledge. The following answers can be proposed from text processing techniques for the research questions guiding the study.

- RQ_1. What are the capabilities and limitations of graph-based representations in capturing syntax and semantics within textual content?
- RQ_3. How do they enable the extraction of both explicit and implicit knowledge?

According to our study, graph-based representations are highly effective in capturing the syntax and semantics of clinical case texts, allowing for the structured representation of complex relationships between medical entities. For example, a graph can represent the relationship between symptoms, diagnoses, treatments, and outcomes in a clinical case by organizing these elements as nodes and their interactions as edges. This structured approach enables the extraction of explicit knowledge, such as the direct link between "chest pain" and "myocardial infarction", and implicit knowledge, such as inferring potential complications based on the connections between related conditions.

Despite their strengths, graph-based representations face challenges in handling the complexity and scale of clinical case data. For instance, as the volume of patient records and associated medical information grows, the resulting graph can become extremely large and complex to manage, leading to performance bottlenecks in storage, processing, and querying. Additionally, the ambiguity inherent in medical language, such as different terms for the same condition or varying interpretations of symptoms, can complicate the construction of accurate text graphs. In a clinical setting, this could result in incorrect relationships being formed, potentially leading to misleading conclusions or missed diagnoses. Moreover, while graphs are effective at representing explicit relationships, they may struggle to fully capture complex semantic nuances, such as the subtleties of patient-reported symptoms or the context-dependent nature of certain medical terms, limiting their ability to model the full scope of clinical knowledge.

- RQ_2. How can a graph-based structural and semantic text representation address missing information, such as incomplete, low-quality, or contradictory annotations, and what are the implications for querying and maintenance?

A graph-based structural and semantic text representation can effectively address missing information, such as incomplete, low-quality, or contradictory annotations, by

leveraging its ability to integrate and connect related data points as nodes through established relationships. When dealing with incomplete or low-quality annotations, graphs can infer missing links by analysing the surrounding context and relationships between known nodes, filling gaps in the data. For instance, if specific medical annotations are incomplete in a clinical case, a graph can use the existing connections between symptoms, diagnoses, and treatments to suggest possible missing links or infer likely associations.

In the case of contradictory annotations, a graph can maintain multiple perspectives by representing conflicting information as alternative paths or nodes within the same structure. This allows for preserving different interpretations or potential errors without discarding any data outright. The implications for querying are significant, as the graph structure enables more flexible and resilient queries that can navigate through incomplete or contradictory data, providing users with a comprehensive view of all possible interpretations. Additionally, for maintenance, graph-based systems can more easily update and refine the structure as new information becomes available, ensuring that the representation remains accurate and relevant. This adaptability makes graph-based models particularly useful in dynamic environments where data quality and completeness may vary.

- RQ_4. What querying possibilities are enabled by structural and semantic text graph representations?

Queries may target factual content by examining terms' probabilistic or definitive presence within textual documents, similar to traditional information retrieval techniques (see Figure 2). Other query types include aggregation operations, such as identifying the most frequently mentioned drugs in a dataset of clinical cases or deducing medical follow-up protocols based on the content of clinical case reports. Additionally, correlating specific symptom patterns with corresponding drug usage to track side effects noted by physicians in clinical records is explored as a practical application of these querying techniques.

Overall, integrating techniques from linguistics, machine learning, and semantic analysis forms the backbone of effective NLP systems, which must continuously evolve to address the intricacies of human language represented through text. Addressing these challenges necessitates innovative solutions that process textual data efficiently and adapt to the dynamic nature of language and its contextual dependencies.

3. Modelling Knowledge from Textual Content: Semantic Web

The Semantic Web extends the World Wide Web by enhancing its intelligence and usability, allowing machines to understand the information available on the Web. A significant challenge in realising this vision is the unstructured nature of textual content, which machines traditionally find difficult to interpret. To address this challenge, text mining techniques are employed to extract meaningful information and semantics from textual data autonomously, facilitating easier machine understanding. Integrating knowledge bases like Wikidata into the Semantic Web framework offers another effective strategy for managing and utilising textual content. These knowledge bases provide structured and readily accessible data that can be easily interpreted by computational methods, thereby enriching the Semantic Web with a deeper layer of context and meaning (https://www.wikidata.org/wiki/Wikidata:Main_Page, accessed on 19 August 2024). These knowledge bases store structured, linked data that can be easily queried and visualised, enhancing machine comprehension of Web content. In the Semantic Web, sentiment analysis [35] also plays a crucial role in interpreting textual content. It involves determining the writer's attitude towards a positive, negative, or neutral topic using natural language processing techniques. This approach not only aids in understanding textual nuances but also enriches the Semantic Web's ability to process and respond to human emotions and opinions expressed online.

The following sections show how to model text content from the Semantic Web approach and how can reasoning can allow to extract and infer knowledge. The section concludes discussing and comparing the modelling of content through ontologies and text graphs and discusses open issues answering the research questions that guide this study.

3.1. Modelling Textual Content

Modelling knowledge from textual content within the Semantic Web framework involves employing ontologies representing explicit, formal, conceptual models. These models semantically describe unstructured content and databases [36]. This methodology has gained substantial traction through initiatives like Schema.org and Linked Open Data (LOD), which facilitate the structured representation and connectivity of data across the Web (https://www.ontotext.com/blog/the-semantic-web-20-years-later/, accessed on 19 August 2024).

The objective of modelling textual content with ontologies is to structure, standardize, and enrich the representation of knowledge within the text, enabling more effective data integration, retrieval, and analysis. Ontologies allow for the organization of textual content into a structured format, where entities, concepts, and their relationships are explicitly defined. This semantic structuring facilitates a deeper understanding of the content, as it captures not just the terms used in the text but also the meanings and relationships among them.

The process of modelling text content with ontologies involves several critical steps [37]:

1. Analysing domain-specific text to identify pertinent terms, concepts, and their relationships;
2. Mapping these elements into an ontology using representation languages such as OWL (Web Ontology Language), RDF (Resource Description Framework), or RDFS (Resource Description Framework Schema);
3. Evaluating the constructed ontology.

Ontology construction can be approached in one of three ways: manual construction, where the ontology is fully crafted by experts; cooperative construction, which requires human intervention during the process; and (semi-) automatic construction, which leverages computational tools to aid in the development of the ontology [38].

Considering the medical treatment of COVID-19, an ontology could be created by extracting key concepts and relationships from the text. For example, treatments, medications, and symptoms could be represented as concepts, and relationships such as "is used to treat", "is a symptom of", or "is a medication for" could be used to connect these concepts. Here is an example of how an ontology for medical treatment of COVID-19 might look:

```
Concepts:   COVID-19, Antiviral Medications,
            Nirmatrelvir with Ritonavir (Paxlovid),
            Remdesivir (Veklury), Fever,
            Cough, Shortness of Breath

Relationships:
Nirmatrelvir with Ritonavir (Paxlovid) is a medication for COVID-19
Remdesivir (Veklury) is a medication for COVID-19
Fever is a symptom of COVID-19
Cough is a symptom of COVID-19
Shortness of Breath is a symptom of COVID-19
```

Relationships (often called properties or relations) are formally defined connections between concepts or entities. These relationships are an integral part of the ontology's structure, contributing to the logical framework that defines how different concepts within a domain are connected.

Ontologies provide a formal and standardized vocabulary for representing concepts and their relationships within a domain. By modelling textual content with ontologies, disparate and unstructured text can be mapped to a consistent set of terms and definitions, making the information more coherent and easier to process.

By structuring textual content with ontologies, it becomes possible to perform semantic searches, where queries can go beyond simple keyword matching to understand the meanings of terms and the relationships between them. This results in more accurate and relevant search results (see Section 3.2).

3.2. Reasoning (Querying) with Ontologies

Ontologies representing text can answer various queries about the domain they represent. One type of query that can be answered using ontologies representing text is conjunctive queries, the basic database queries [39]. These are existentially quantified conjunctions of atoms, meaning they return all combinations of values that satisfy the conditions specified in the query. Conjunctive queries can retrieve information from an ontology by selecting the desired relationships between concepts.

Text graphs and ontologies serve complementary roles in the representation and analysis of textual content. In text graphs, relationships (often called edges) are derived directly from the text. These relationships represent the connections between entities (nodes) based on how they are mentioned or inferred from the text. The creation of these edges is typically based on co-occurrence, dependency parsing, or semantic analysis of the text. Relationships in text graphs are highly contextual, reflecting the specific instances of interaction or association found in the text. For example, "John treated Mary" or "COVID-19 causes fever". These relationships are extracted directly from sentences or passages in the text.

In ontologies, relationships (often called properties or relations) are formally defined connections between concepts or entities. Ontological relationships can be hierarchical, associative, logical constraints.

- Hierarchical relationships include subclass or subtype relationships (e.g., "A dog is a mammal"). They establish a hierarchy within the ontology, helping to organise concepts into categories and subcategories.
- Associative relations connect entities in a non-hierarchical manner, such as "A doctor treats a patient" or "A drug is prescribed for a disease". They define how different concepts interact or are related within the domain.
- Ontological relationships often come with logical constraints or axioms (e.g., inverse, transitive, symmetric properties). These constraints help in reasoning and inference, enabling the derivation of new knowledge from existing facts.

One way to query ontologies is to use SPARQL (SPARQL is a semantic query language for databases able to retrieve and manipulate data stored in Resource Description Framework (RDF) format https://en.wikipedia.org/wiki/SPARQL, accessed on 19 August 2024), a query language for RDF data (The Resource Description Framework (RDF) is a World Wide Web Consortium (W3C) standard designed as a data model for metadata. It has come to be used as a general method for description and exchange of graph data). SPARQL allows users to write queries that retrieve and manipulate data stored in RDF format. This can include data stored in triplestores, a database specifically designed for storing and querying RDF data.

Another approach to querying ontologies is using reasoners, which can infer new knowledge from the existing data based on the logical rules defined in the ontology. This can allow for more complex queries considering the relationships between different entities and their properties. There are various tools and frameworks available for querying ontologies like SPARQL, Protégé (https://protege.stanford.edu/, accessed on 19 August 2024) and Apache Jena (https://learn.microsoft.com/en-us/azure/architecture/data-science-process/platforms-and-tools, accessed on 19 August 2024).

Based on the ontology above, some example queries that could be used to retrieve information from it are:

```
What are the medications for COVID-19?
Query:
SELECT ?x WHERE { ?x is a medication for COVID-19 }
Result: Nirmatrelvir with Ritonavir (Paxlovid), Remdesivir (Veklury)

What are the symptoms of COVID-19?
Query: SELECT ?x WHERE { ?x is a symptom of COVID-19 }
```

```
Result: Fever, Cough, Shortness of Breath
```

The Clinical MetaData Ontology (CMDO) (https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-019-0877-x, accessed on 19 August 2024) is an example of an ontology representing clinical cases. It provides a simple classification scheme for clinical data elements based on semantics and comprises 189 concepts under four first-level classes. Below are some SPARQL query examples that demonstrate how you can reason over a clinical dataset modelled with CMDO.

- Querying basic information: Retrieve basic information about all patients in the dataset.

```
PREFIX cmdo: <http://example.org/cmdo#>

SELECT ?patient ?name ?age ?gender
WHERE {
        ?patient a cmdo:Patient ;
        cmdo:hasName ?name ;
        cmdo:hasAge ?age ;
        cmdo:hasGender ?gender .
    }
```

- Finding patients with specific conditions: Retrieve the list of patients diagnosed with a specific condition, such as diabetes.

```
PREFIX cmdo: <http://example.org/cmdo#>

SELECT ?patient ?name
WHERE {
  ?patient a cmdo:Patient ;
           cmdo:hasName ?name ;
           cmdo:hasCondition ?condition .
  ?condition a cmdo:Condition ;
             cmdo:hasName "Diabetes" .
}
```

- Reasoning over treatment outcomes: Identify patients who were treated with a specific medication and their associated treatment outcomes.

```
PREFIX cmdo: <http://example.org/cmdo#>

SELECT ?patient ?name ?medication ?outcome
WHERE {
  ?patient a cmdo:Patient ;
           cmdo:hasName ?name ;
           cmdo:receivedTreatment ?treatment .
  ?treatment a cmdo:Treatment ;
             cmdo:usesMedication ?medication ;
             cmdo:hasOutcome ?outcome .
  ?medication a cmdo:Medication ;
              cmdo:hasName "Metformin" .
}
```

- Identifying potential risk factors: Identify patients with multiple risk factors (e.g., high blood pressure and high cholesterol) who have not yet been diagnosed with a related condition (e.g., cardiovascular disease).

```
PREFIX cmdo: <http://example.org/cmdo#>

SELECT ?patient ?name
WHERE {
  ?patient a cmdo:Patient ;
           cmdo:hasName ?name ;
           cmdo:hasRiskFactor ?riskFactor1, ?riskFactor2 .
  ?riskFactor1 a cmdo:RiskFactor ;
               cmdo:hasName "High Blood Pressure" .
  ?riskFactor2 a cmdo:RiskFactor ;
               cmdo:hasName "High Cholesterol" .
  FILTER NOT EXISTS {
    ?patient cmdo:hasCondition ?condition .
    ?condition a cmdo:Condition ;
               cmdo:hasName "Cardiovascular Disease" .
  }
}
```

- Detecting condition progression: Detect patients whose condition has progressed based on lab results or other clinical measurements.

```
    PREFIX cmdo: <http://example.org/cmdo#>

SELECT ?patient ?name ?condition ?measurement ?value ?date
WHERE {
  ?patient a cmdo:Patient ;
           cmdo:hasName ?name ;
           cmdo:hasCondition ?condition ;
           cmdo:hasMeasurement ?measurementRecord .
  ?condition a cmdo:Condition ;
             cmdo:hasName "Diabetes" .
  ?measurementRecord a cmdo:Measurement ;
                     cmdo:hasMeasurementType ?measurement ;
                     cmdo:hasMeasurementValue ?value ;
                     cmdo:hasMeasurementDate ?date .
  FILTER (?measurement = "HbA1c" && ?value > 7.0)
}
```

- Inferring new knowledge through reasoning: Infer patients at risk of a condition based on related factors and existing conditions, applying reasoning over the ontology.

```
    PREFIX cmdo: <http://example.org/cmdo#>

CONSTRUCT {
  ?patient cmdo:atRiskOf ?condition .
}
WHERE {
  ?patient a cmdo:Patient ;
           cmdo:hasCondition ?existingCondition ;
           cmdo:hasRiskFactor ?riskFactor .
  ?existingCondition a cmdo:Condition ;
                     cmdo:isRelatedTo ?condition .
  ?condition a cmdo:Condition .
}
```

- Finding treatment patterns: Identify common treatment patterns for a particular condition across multiple patients.

```
    PREFIX cmdo: <http://example.org/cmdo#>

SELECT ?condition ?treatment (COUNT(?patient) AS ?numPatients)
WHERE {
  ?patient a cmdo:Patient ;
           cmdo:hasCondition ?condition ;
           cmdo:receivedTreatment ?treatment .
  ?condition a cmdo:Condition ;
             cmdo:hasName "Hypertension" .
}
GROUP BY ?condition ?treatment
ORDER BY DESC(?numPatients)
```

- Predicting complications: Predict potential complications for patients with a specific condition based on past data.

```
    PREFIX cmdo: <http://example.org/cmdo#>

SELECT ?patient ?name ?predictedComplication
WHERE {
  ?patient a cmdo:Patient ;
           cmdo:hasName ?name ;
           cmdo:hasCondition ?condition ;
           cmdo:hasRiskFactor ?riskFactor .
  ?condition a cmdo:Condition ;
             cmdo:hasName "Diabetes" .
  ?riskFactor a cmdo:RiskFactor ;
              cmdo:predictsComplication ?predictedComplication .
}
```

These SPARQL queries demonstrate how reasoning over a Clinical MetaData Ontology (CMDO) can provide deep insights into clinical data, support decision-making, and predict patient outcomes. Reasoning on text ontologies involves deriving facts not explicitly expressed in the ontology or knowledge base. Reasoning with ontologies is becoming increasingly crucial in data-centric applications, where it helps to integrate heterogeneous data and provide a common conceptual vocabulary.

3.3. Discussion: Current Trends and Open Issues

The following section discusses conclusions on how the Semantic Web can be leveraged to model textual content using ontologies, enabling advanced querying and reasoning to extract meaningful knowledge.

This study explores various approaches, including the Semantic Web, for modelling textual content and extracting knowledge, with the goal of identifying current trends and addressing open issues. These aspects are examined through the lens of four key research questions that guide the analysis.

- RQ_1. What are the capabilities and limitations of ontology-based representations in capturing syntax and semantics within textual content?

One of the main challenges is the difficulty of automatically extracting meaningful information and semantics from unstructured text. This process, known as ontology learning, involves using techniques from machine learning, text mining, knowledge representation and reasoning, information retrieval, and natural language processing to acquire ontologies from unstructured text automatically [40]. NLP techniques can automatically extract meaningful information and semantics from text, which can be represented using Semantic

Web technologies such as RDF and OWL. This allows for creating structured, linked data that can be easily queried and manipulated using Semantic Web query languages such as SPARQL [41].

Another challenge is the ambiguity and complexity of natural language. Natural language labels used to describe the contents of hierarchical classifications are easily understood by human users but can be difficult for machines to reason about due to their ambiguity [42]. This creates a challenge for converting classifications into lightweight ontologies that can be used in the Semantic Web infrastructure. In addition, building large ontologies is difficult and time-consuming, and it is not feasible to build ontologies for all available domains. Therefore, it is necessary to develop automated methods for ontology learning that can scale to handle large amounts of data.

- RQ_2. How can an ontology-based structural and semantic text representation address missing information, such as incomplete, low-quality, or contradictory annotations, and what are the implications for querying and maintenance?

One approach to address this issue is to use quality assurance techniques to improve the quality of the ontology [43]. For example, in one study in [44], the detection of semantic regularities in entailed axioms can be used in ontology quality assurance in combination with lexical techniques. Another approach is to use quality evaluation methodologies to assess the quality of the ontology and identify areas for improvement [45]. Additionally, tools and frameworks can help with the documentation and management of quality assurance measures for ontologies [46].

Incomplete data in ontologies can significantly impact querying and maintenance. When data is incomplete, it can be difficult to answer queries accurately or make inferences based on the available information. This can lead to incorrect or incomplete results, which can affect the usefulness and reliability of the ontology.

Several approaches have been developed to address these issues. One approach uses ontology-based data access (OBDA) techniques to access incomplete and/or heterogeneous data [47]. These techniques allow ontologies to provide taxonomies and background knowledge to help align and complete the data.

- RQ_3. How do ontologies enable the extraction of both explicit and implicit knowledge?

Ontology evolution techniques can detect ontological changes and update the annotations accordingly. The work in [48] used annotators to generate more than 66 million annotations from a pre-selected set of 5000 biomedical journal articles and standard ontologies covering a period ranging from 2004 to 2016. The work highlighted the correlation between ontological and annotation changes and discussed the necessity of improving existing annotation formalisms to include elements required to support (semi-) automatic annotation maintenance mechanisms.

Another approach is to use ontology alignment techniques to maintain the consistency of annotations across different ontologies [49,50]. It can preserve the validity of ontology alignment by only analysing changes introduced to maintained ontologies. The precise definition of ontologies is provided, along with a definition of the ontology change log. A set of algorithms that allow revalidating ontology alignments have been built based on such elements [49].

Validating new knowledge in an ontology can be done using a validation and verification-driven ontology design process, such as the CQ-Driven Ontology Design Process (CODEP) proposed by Espinoza et al. [51]. This process is driven by end-user requirements, defined as Competency Questions (CQs). It includes activities that validate and verify the incremental design of an ontology through metrics based on defined CQs. Another approach is to use a method to validate the insertion of a new concept in an ontology, such as the one presented by [52]. This method is based on finding the neighbourhood of the concept in the ontology and assessing its semantic similarity with its neighbourhood in a general ontology, then evaluating the correlation between the values found in these steps. Several tools are available for ontology evaluation and validation, such as OntoQA [53], which

uses a set of metrics to measure different aspects of the ontology schema and knowledge base to give insight into the overall characteristics of the ontology.

- RQ_4. What querying possibilities are enabled by structural and semantic text ontologies representations?

NLP techniques such as named entity recognition, relation extraction, and sentiment analysis can enhance the querying of text content by allowing for more sophisticated queries that consider the text's meaning and context. For example, a query could be constructed to find all documents that mention a specific person or organisation or to find all documents that express a positive sentiment towards a particular topic [41].

Relationships in ontologies facilitate complex queries that can leverage the logical structure to answer questions that require understanding both direct and inferred relationships. For instance, querying an ontology for "All treatments for cardiovascular diseases" might automatically include treatments for specific conditions like hypertension or heart attacks, due to the hierarchical and associative relationships defined in the ontology.

Relationships in text graphs support exploratory queries that seek to uncover relationships or connections that are not explicitly defined in a structured format. For example, querying a text graph for "entities related to COVID-19" might return a diverse set of nodes and edges representing symptoms, treatments, and societal impacts, all connected by various contextual relationships derived from different documents.

Text graphs are more flexible and adaptable, making them ideal for exploratory analysis and dynamic applications where relationships need to be discovered and updated in real-time.

Ontologies, on the other hand, provide a stable and structured framework for consistent knowledge representation, making them essential for data integration, interoperability, and reasoning in domains requiring standardized semantics.

Depending on the goals and requirements of a project, one may choose to use text graphs, ontologies, or a combination of both to achieve the desired outcomes. While ontological relationships provide a rigorous framework for understanding and reasoning about knowledge, text graph relationships offer a more fluid and adaptable approach to uncovering and analysing connections within textual content.

Reasoners and SPARQL query engines can be used with machine learning but do not have built-in machine learning functions. SPARQL-extended query language where the user can include a 'function' concerning graph data analysis can be relevant. One example of such an extension is the addition of recursive features to the SPARQL query language [54], which allows for expressing analytical tasks over existing SPARQL infrastructure. This language is well-suited for graph querying and analytical tasks and can express key analytical tasks on graphs [55].

The advantages of such an extension include combining querying and analytics for graphs, allowing users to perform complex analytical tasks using a single query language. This can simplify the querying and analysing of graph data, making it easier for users to extract meaningful insights from their data [56]. Some uses of this extended query language could include performing complex graph analytics, such as finding the shortest path between two nodes, computing centrality measures, or identifying clusters or communities within the graph [57].

One challenge in developing such an extension is ensuring it is compatible with existing SPARQL infrastructure and can be easily integrated into existing systems. Another challenge is ensuring that the language is expressive enough to support a wide range of analytical tasks while remaining easy to use and understand [58].

4. Graph Databases for Storing, Querying and Analysing Textual Content

A graph database management system (GDBMS) that uses graph structures, including nodes, edges, and properties, to represent and store data. The fundamental concept of a graph database is the graph, which relates data items in the store to a collection of nodes and edges, with the edges representing the relationships between the nodes. GDBMSs

relay on a graph data model that is the data structure used for structuring content to be efficiently managed (i.e., stored, updated, queried). There are two popular models used for designing graph databases: property graphs and RDF graphs. Property graphs focus on analytics and querying, while RDF graphs emphasise data integration. Both graph models consist of nodes (vertices) and the connections between those points (edges). In this study we focus mainly on property graphs. Graph databases prioritise relationships between data, making querying relationships fast and efficient. Relationships can also be intuitively visualised using graph databases, making them useful for managing and querying heavily inter-connected data.

Graph databases are well adapted for managing data with a graph structure, for example, textual content, networks (social, transport, biology, etc.), and financial transactions. The following lines give an overview of graph data models and how they can be used for modelling textual content. Then they discuss the interest of persistence in the process of structuring textual content and ultimately extracting and maintaining knowledge. Finally, this section shows how text graph databases can be explored, queried and analysed for extracting knowledge.

4.1. Building Graph Databases

Building a graph database from a text document dataset can be approached in two primary ways: with a schema a priori definition or a schemaless approach (bottom-up).

In the schemaless approach, data is ingested and transformed into a graph dynamically, with nodes, edges, and properties created as the data is processed. This bottom-up method allows the graph to evolve organically, accommodating a wide range of data types and structures without the need for predefined constraints. While this flexibility is advantageous for handling diverse and unpredictable data, it can also lead to challenges in maintaining consistency and efficiency in querying.

Conversely, the schema a priori approach involves defining a fixed schema before populating the graph. This top-down method requires careful planning to establish the types of nodes, relationships, and properties expected in the dataset. This structured approach ensures consistency and facilitates optimised querying but can be less adaptable when new or unexpected data types are encountered, requiring potential schema revisions.

The schema a priori approach offers stability and predictability, ideal for well-understood domains, while the schemaless approach provides the flexibility needed to handle dynamic and evolving datasets, albeit with potential trade-offs in complexity and consistency.

Graph Database Modelling

Graph database modelling translates a conceptual view of data (top-down) or data entities (bottom-up) into a logical model (e.g., property graphs). In a graph database, nodes represent the fundamental data units, while edges represent the relationships between nodes. Properties are descriptive characteristics of nodes and edges that are not important enough to become nodes themselves.

There is no formula for deriving a graph model from a dataset, but there are general guidelines that can help create an effective model. For example, it is essential to consider the types of questions to answer with data and design the graph database accordingly (https://cambridge-intelligence.com/graph-data-modeling-101/, accessed on 19 August 2024) [59]. Therefore, decision-making must determine which entities in a dataset should be nodes, which should be relationships, which should be properties and which should be discarded (!). The result is a blueprint of data's entities, relationships, and properties.

Consider for instance, a 55-year-old male patient presents with chest pain and shortness of breath. The patient has a history of hypertension and diabetes. Upon examination, the electrocardiogram (ECG) indicates signs of a myocardial infarction (MI). The patient is prescribed aspirin and statins. Follow-up tests reveal elevated cholesterol levels, and the patient is advised to continue with the prescribed medications and adopt a healthier lifestyle.

[1] Decision-Making Challenge in Transforming into a Property Graph. When transforming this clinical case text into a property graph, decisions must be made regarding which entities are modelled as nodes, properties, and edges. Deciding what to model as nodes, properties, and edges involves considering the importance and relationships of entities within the clinical context:

- Nodes: Entities central to the clinical decision-making process (e.g., patient, conditions, and medications) are modelled as nodes because they represent key concepts or actors in the clinical narrative.
- Properties: Attributes that describe these entities without requiring separate nodes are modelled as properties. For instance, the patient's age and gender are properties of the patient node rather than separate nodes because they describe the patient rather than act as independent entities.
- Edges: Relationships critical to understanding the flow of the clinical case are modelled as edges. For instance, the connection between a diagnosis and its treatment is best represented as an edge to clearly show the therapeutic decision process.

[2] Graph database modelling challenges:

- Ambiguity: Deciding whether to model "elevated cholesterol levels" as a property of a test result or as a separate node connected by an edge can be ambiguous and depends on the specific use case and the granularity of the graph.
- Complexity: The more detailed the graph, the more complex it becomes. For instance, every medication might have dosage, frequency, and side effects as properties, but modelling these might complicate the graph unnecessarily for certain queries.
- Consistency: Ensuring that similar entities across different patient records are consistently modelled is crucial for making the graph queryable and comparable across cases.

The choices made in modelling can impact how easily the graph can be queried for insights. A well-structured graph allows for efficient querying, such as finding all patients with similar symptoms and conditions, or analysing the effectiveness of treatments across cases. Poor modelling, on the other hand, can lead to difficulties in extracting useful information and maintaining the graph over time as new data is added.

4.2. Defining and Querying a Graph Database

Graph database management systems (GDBMSs) offer mechanisms for both defining and querying graphs, which are essential for efficiently managing and analysing complex, interconnected data. These systems allow users to define the structure of the graph through schema definition or schema-less approaches, specifying the types of nodes, relationships, and properties that will be used to model the data. Additionally, GDBMSs provide declarative query languages, such as Cypher (Cypher is a declarative graph query language that allows for expressive and efficient data querying in a property graph) or GQL (GQL (graph query language) is a standardized query language for property graphs first described in ISO/IEC 39075:2024, released in April 2024 by ISO/IEC https://www.iso.org/standard/76120.html accessed on 19 August 2024), that enable users to perform queries on the graph data.

4.2.1. Defining a Graph Database

GDBMSs implement a data definition language (DDL) for allowing the declarative definition of a graph database. The DDL plays a foundational role in establishing the schema and structural integrity of a graph database. It ensures that the graph data is organised, optimised, and consistent, allowing for efficient querying, maintenance, and evolution of the database over time. Through DDL, a graph database becomes a robust and reliable platform for managing complex and interconnected data.

The following Cypher code defines a co-occurrence graph, assuming the representation of documents containing clinical cases. Nodes are created with the label "Word" and

a property name which holds the word itself. Relationships are created with a type "CO_OCCURS_WITH" and a property "weight" that quantifies the frequency or strength of the co-occurrence. Each node represents a unique word found in clinical reports. In this example, the words are "Fever", "Cough", and "Influenza". The edges between nodes represent the co-occurrence of two words within the same document or a specified context (like a sentence or paragraph). This relationship is not just a binary indicator; it also includes a weight attribute, quantifying the co-occurrence's strength or frequency. For instance, a weight of 15 between "Fever" and "Cough" suggests a strong or frequent co-occurrence, indicating that these words often appear together, which could imply a common association in clinical symptoms or conditions. Lower weights, such as 5 between "Cough" and "Influenza", indicate less frequent co-occurrences. This setup allows you to query the graph to determine how often words co-occur, explore the relationships between different terms, and extend the model to include additional attributes or relationships as needed.

```
CREATE
  (fever:Word {name: 'Fever'}),
  (cough:Word {name: 'Cough'}),
  (influenza:Word {name: 'Influenza'}),
  (fever)-[:CO_OCCURS_WITH {weight: 15}]->(cough),
  (fever)-[:CO_OCCURS_WITH {weight: 10}]->(influenza),
  (cough)-[:CO_OCCURS_WITH {weight: 5}]->(influenza)
```

The challenge lies in transforming textual content into graphs that accurately capture the content's meaning. From a database design perspective, determining the most appropriate graph models for designing these graphs, such as property graphs, directed acyclic graphs (DAGs), undirected acyclic graphs (unDAGs), etc., is crucial.

4.2.2. Querying and Analysing a Graph Database

Textual content representation as graphs allows to perform traversal queries based on connections, use specific algorithms on graphs, find patterns, paths, communities, influencers, single points of failure, and other relationships, and have a more intuitive understanding and visualisation of the data content.

The principle of querying a graph database revolves around the traversal of nodes and edges to discover patterns, relationships, and properties within the graph. Queries in a graph database are designed to explore the connections between these nodes, allowing users to retrieve information based on the direct or indirect relationships that exist within the data. One could use a query language like Cypher, GQL or Gremlin (Gremlin is a graph traversal language and virtual machine developed by Apache TinkerPop of the Apache Software Foundation) to retrieve information from the graph.

Graph algorithms are analytic tools used to determine the strength and direction of relationships between objects in a graph. Machine learning is a powerful tool for addressing various problems and gaining new insights from graphs. Some typical problems that can be addressed with machine learning and graphs include recommendation systems, community detection, link prediction, and node classification. Graphs can be used to represent the relationships between users and items, and machine learning algorithms can be applied to these graphs to:

- make personalised recommendations;
- identify clusters or communities of nodes that are more densely connected than to the rest of the graph;
- predicts the likelihood of a link forming between two nodes in a graph based on the characteristics of the nodes and their relationships with other nodes in the graph;
- classify the nodes in a graph based on their attributes and relationships with other nodes.

Graphs can be ingested into machine learning algorithms and then be used to perform classification, clustering, regression, etc. Together, graph and machine learning provide

greater analytical accuracy and faster insights. For example, graph features can work as an input for machine learning, like using PageRank score as a feature in a machine learning model. Machine learning can also enhance graphs, for example, performing entity resolution (Entity resolution is the process that resolves entities and detects relationships. The pipelines perform entity resolution as they process incoming identity records in three phases: recognise, resolve, and relate) and combine different data sources into one single graph (https://www.ibm.com/docs/en/iii/9.0.0?topic=insight-entity-resolution accessed on 19 August 2024) [60,61].

[1] Analysing text graph databases through an example: a medical graph database.

The following example demonstrates how to use Neo4j's built-in machine learning capabilities to build, train, and deploy predictive models directly within a medical cases graph. Consider a large dataset of medical cases, including patient information, diagnoses, treatments, outcomes, and associated clinical notes. Our goal is to use this data to predict the likelihood of a patient developing a particular condition (e.g., diabetes) based on their medical history and current symptoms. We use for this example Neo4J, a graph database management system (GDBMS) which includes built-in machine learning models, to create, analyse, and derive insights from a graph representing these medical cases.

We define a graph database consisting of the following:

- node types representing entities like patient, condition, symptom, medication, and treatment
- edges like
  Patient -> condition (diagnosed_with);
  Patient -> medication (treated_with);
  Condition -> symptom (exhibits);
  Treatment -> outcome (results_in).

The graph database represents a structured medical case data model. Assume that it contains two patients "John Doe", a 45-year-old male, and "Jane Smith", a 50-year-old female. The conditions represented in the cases are hypertension and diabetes. Symptoms such as fatigue and thirst, and medications like metformin and amlodipine, are added as distinct nodes. The database establishes relationships between these entities to reflect real-world medical scenarios. For example, "John Doe" is diagnosed with hypertension and treated with amlodipine, while diabetes is associated with the symptom thirst. We can use Neo4j's built-in support for logistic regression to predict the likelihood of developing diabetes.

```
CALL gds.alpha.ml.train.logisticRegression({
    nodeLabels: ['Patient'],
    featureProperties: ['age', 'avgConditionPageRank'],
    targetProperty: 'hasDiabetes',
    relationshipType: 'DIAGNOSED_WITH'
})
YIELD modelInfo, metrics
RETURN modelInfo, metrics;
```

The model performance can then be evaluated using the output from the above query.

```
CALL gds.alpha.ml.evaluateModel('logisticRegression')
YIELD accuracy, f1Score, rocAuc, precision, recall
RETURN accuracy, f1Score, rocAuc, precision, recall;
```

Once the model is trained, it can be possible to make real-time predictions for new patients.

```
MATCH (p:Patient {id: 'P3'})
WITH p, gds.alpha.ml.predict('logisticRegression', p) AS diabetesPrediction
RETURN p.name, diabetesPrediction;
```

The prediction can be integrated with decision support. For example, suggest lifestyle changes or further tests if the model predicts a high likelihood of diabetes.

```
MATCH (p:Patient)-[:TREATED_WITH]->(m:Medication)
WHERE p.diabetesPrediction > 0.8
RETURN p.name, m.name AS recommendedMedication;
```

As new data arrives, update the graph and retrain the model periodically.

```
// Add new patient data
CREATE (:Patient {id: 'P4', name: 'Alice Johnson', age: 48, gender: 'F'});

// Retrain the model with updated data
CALL gds.alpha.ml.train.logisticRegression({
  nodeLabels: ['Patient'],
  featureProperties: ['age', 'avgConditionPageRank'],
  targetProperty: 'hasDiabetes',
  relationshipType: 'DIAGNOSED_WITH'
})
YIELD modelInfo, metrics
RETURN modelInfo, metrics;
```

For monitoring and maintenance it is possible to continuously monitor the model performance and make adjustments as necessary.

```
CALL gds.alpha.ml.monitor('logisticRegression')
YIELD accuracy, rocAuc
RETURN accuracy, rocAuc;
```

This example demonstrates how to use Neo4j's built-in machine learning capabilities to build, train, and deploy predictive models directly within a medical cases graph. By leveraging the combination of graph algorithms, feature engineering, and real-time prediction, this approach provides insights for healthcare providers, enabling more accurate diagnoses and effective treatments.

[2] Graph analytics in GDBMS. A GDBMS and a graph data science platform can be utilised to calculate various graph metrics, including generic metrics such as betweenness centrality [62] and PageRank [16,63] score, as well as domain-specific metrics such as the number of known fraudsters indirectly connected to a given client. Additionally, these tools can generate graph embeddings [64], translating graph data into a more suitable format for machine learning.

Many graph databases, including Amazon Neptune, OrientDB, ArangoDB, JanusGraph, and TigerGraph, can analyse graphs using machine learning algorithms [65]. Graph-based DBMSs offer a variety of analytics and data science solutions for processing graphs. These solutions include using platforms for enterprise knowledge graphs or graph artificial intelligence (AI) and graph-based deep learning approaches to improve analytics. The graph DBMS market vendors are expanding their stacks into platforms for enterprise knowledge graphs or graph AI with associated product ecosystems. For example, graph DBMS can use a graph convolution network (GCN) to enable CRM analytics to forecast sales.

The Neo4j graph data science add-on provides a powerful set of tools for analysing relationships in data using graph algorithms and machine learning. The library includes a catalogue of 65+ graph algorithms, graph native ML pipelines, and graph visualisation tools, allowing users to answer questions such as what is important, what is unusual, and what is next.

Amazon Neptune has a feature called Neptune ML that uses graph neural networks (GNNs) to make predictions using graph data. Neptune ML can improve the accuracy of most predictions for graphs by over 50% when compared to making predictions using

non-graph methods (https://aws.amazon.com/neptune/machine-learning/, accessed on 19 August 2024).

OrientDB has a suggestion algorithm that leverages graph-based data modelling and machine learning techniques to generate intelligent recommendations (https://saturncloud.io/blog/orientdb-suggestion-algorithm-enhancing-data-analysis-with-intelligent-recommendations/, accessed on 19 August 2024).

ArangoDB has a graph data science library that includes over 50 algorithms for dependencies, clustering, similarity, matching/patterns, flow, centrality, and search (https://www.arangodb.com/graph-analytics-at-enterprise-scale/, accessed on 19 August 2024). Many machine learning algorithms are based on graphs.

JanusGraph supports graph processing. Scaling graph data processing for real-time traversals and analytical queries is JanusGraph's foundational benefit (https://docs.janusgraph.org/, accessed on 19 August 2024).

TigerGraph has an in-database graph data science library that includes over 50 algorithms for dependencies, clustering, similarity, matching/patterns, flow, centrality, and search (https://www.tigergraph.com/graph-data-science-library/, accessed on 19 August 2024).

[3] Using a GDBMS for graph analytics: concluding insights. Using a graph database management system (GDBMS) for performing graph analytics offers several significant advantages over traditional approaches based on programming languages and machine learning libraries.

GDBMSs store data in a graph-native format where nodes and relationships are first-class citizens, inherently supporting efficient data traversal and querying. This contrasts with relational databases or flat-file storage, which require complex joins or custom data structures to represent graphs, often leading to slower performance and increased complexity. For example, in a medical cases database, querying the relationships between patients, symptoms, and treatments is much faster and more intuitive in a GDBMS, enabling quicker insights into patient care.

GDBMSs also feature declarative query languages like Cypher and GSQL, specifically designed for graph data manipulation. These languages allow users to perform sophisticated analytics through succinct, high-level queries without the need to write procedural code, reducing both the learning curve and development time. In the context of medical data, this means healthcare providers can quickly generate complex queries, such as finding all patients with similar symptoms and treatment outcomes, without needing in-depth programming knowledge.

Furthermore, GDBMSs come equipped with built-in graph algorithms, such as PageRank and community detection, optimised for the database's architecture. These algorithms can be applied directly to the data stored within the graph, enabling scalable analytics even as the graph grows in size and complexity.

One of the most powerful features of GDBMSs is the ability to define graph views—subsets or projections of the graph tailored to specific analytical tasks. In a medical graph, one might create a view focusing on the relationships between diseases and symptoms, while another might highlight treatment outcomes for specific demographics. These views allow for dynamic analysis, enabling users to explore different perspectives of the same underlying data without altering the original dataset. This flexibility is crucial in healthcare, where different stakeholders may need to analyse the same data from various angles.

Additionally, GDBMSs enable real-time analysis and decision support, allowing for instantaneous queries that provide immediate insights. For example, during a patient consultation, a healthcare provider could query the database to find similar cases and recommended treatments on-the-fly, enhancing the decision-making process. GDBMSs can also be integrated into real-time decision support systems, where analytics results directly influence clinical actions—a capability that is more challenging to achieve with stand-alone programming languages and libraries.

Finally, GDBMSs ensure data consistency and integrity through their adherence to ACID properties, automatically maintaining the graph's consistency as new data is added or existing data is updated. This simplifies maintenance and reduces the risk of errors, which is particularly important in domains like healthcare where data accuracy is critical. In contrast, custom-built solutions using traditional programming languages require significant effort to enforce consistency and accommodate changes in the data or analytics requirements, making GDBMSs a more efficient and reliable choice for managing complex, evolving datasets.

By leveraging the built-in capabilities of a GDBMS, users can perform graph analytics more efficiently, with greater flexibility and scalability, than with traditional programming-based approaches. This leads to more timely insights, reduced development overhead, and the ability to adapt quickly to new analytical challenges.

4.2.3. Schema Inference

Frequently, graph data comes without a predefined structure and in a constraint-less fashion, thus leading to inconsistency and poor quality. Inferring the schema can allow understanding the connections in the data better, maintaining its quality and exploiting graphs through navigational and AI-based queries.

Schema inference in a graph database involves deducing the structure or schema of the graph (e.g., node types, edge types, and properties) based on the data already present in the database. Extracting a schema from property graph instances can be challenging and complex, mainly when the instances exist before the schema definition [66].

Neo4j provides a limited ability to view the schema of a database through a Cypher query that outputs a single-labelled directed graph. This graph displays the types of nodes that can be connected and the types of edges that can connect them. There are some limitations to viewing the schema of a database in Neo4j. For example, multi-labelled nodes in the graph instance are duplicated in the graph schema so that each node is assigned a single label. This results in the loss of label co-occurrence information, a critical property graph data model feature. Additionally, properties, edge cardinality constraints, and node type hierarchies are not considered.

Graph query language (GQL) can be used to explore and infer this schema by querying the graph's metadata. For example, consider a graph database where medical cases are represented. Suppose we have nodes representing patients, conditions, medications, and doctors, with edges like diagnosed_with, prescribed, and treated_by. The following steps can be performed to infer the schema based on the existing data:

[1] Inferring Node Types and Their Properties. Query the graph to identify the types of nodes and the properties associated with each type. The following expression gives an example on how to implement this task. The query MATCH (n) is used to match all nodes in the graph, allowing to access and analyse each node individually. The function labels(n) retrieves the labels assigned to these nodes, which typically indicate the type of the node, such as patient or condition. The function keys(n) is then used to extract the properties associated with each node, revealing the attributes or data fields that describe the node (e.g., age, name, severity). The DISTINCT keyword ensures that the results returned are unique, meaning that each node type and its associated properties are only listed once, avoiding duplicates and providing a clear overview of the graph's structure.

```
MATCH (n)
RETURN DISTINCT labels(n) AS NodeTypes, keys(n) AS Properties
ORDER BY NodeTypes;
```

Example result:

```
RelationshipType  | FromNodeType   | ToNodeType
------------------|----------------|----------------
```

```
"diagnosed_with"  | ["Patient"]    | ["Condition"]
"prescribed"      | ["Doctor"]     | ["Medication"]
"treated_by"      | ["Patient"]    | ["Doctor"]
```

[2] Inferring Relationships Between Node Types. Query the graph to infer the relationships (edges) between these node types. The query MATCH (a)-[r]->(b) is designed to match all relationships in the graph that connect one node to another, allowing to examine how different entities in the graph are related. The type(r) function is then used to retrieve the type of each relationship, such as diagnosed_with or prescribed, which describes the nature of the connection between the nodes. Additionally, labels(a) and labels(b) are used to retrieve the labels of the nodes at each end of the relationship, indicating the types of entities that are being connected (e.g., a patient node connected to a condition node). This query helps in understanding the structure and semantics of the relationships within the graph.

```
MATCH (a)-[r]->(b)
RETURN DISTINCT type(r) AS RelationshipType, labels(a) AS
From NodeType, labels(b) AS ToNodeType
ORDER BY RelationshipType;
```

Example result:

```
RelationshipType  | FromNodeType   | ToNodeType
------------------|----------------|----------------
"diagnosed_with"  | ["Patient"]    | ["Condition"]
"prescribed"      | ["Doctor"]     | ["Medication"]
"treated_by"      | ["Patient"]    | ["Doctor"]
```

[3] Inferring Property Types and Values. To understand the kind of data each property holds, query for examples of property values. A function from the APOC library returns the type of the property (e.g., String, Integer, Float).

```
MATCH (n)
RETURN DISTINCT labels(n) AS NodeType, keys(n) AS Property,
apoc.meta.type(n[keys(n)[0]]) AS PropertyType
ORDER BY NodeType, Property;
```

Example result:

```
NodeType      | Property   | PropertyType
--------------|------------|--------------
["Patient"]   | "name"     | "String"
["Patient"]   | "age"      | "Integer"
["Medication"]| "dosage"   | "String"
["Condition"] | "severity" | "String"
```

Through these GQL queries, the inferred schema for the medical case graph database defines:

- Node Types: patient, condition, medication, doctor.
- Relationships: diagnosed_with (patient -> condition), prescribed (doctor -> medication), treated_by (patient -> doctor).
- Properties: Nodes like patient have properties such as name, age, and gender, with types such as String and Integer.

This inferred schema can be used to structure future data insertions, ensure consistency in the data model, or optimise querying strategies within the medical cases graph database.

GraphQL schemas can be derived from Neo4j databases using tools such as the Neo4j Desktop GraphQL plug-in or neo4j-graphql-js. These tools can infer node and edge types

and the data types of node properties. However, unlike other methods, they do not infer overlapping types, node hierarchies, or nested property values. Many approaches to schema inference involve identifying structural graph summaries by grouping equivalent nodes. Clustering techniques, such as those described in [67,68], are commonly used to infer types in RDF datasets. These techniques are based on the assumption that the more properties two entities share, the more likely they are to belong to the same type. To achieve this, entities are grouped according to a similarity metric. Both techniques can handle hierarchical and overlapping types. In [68], a density-based clustering method called DBSCAN is used. In contrast, ref. [67] proposes a faster and more accurate clustering method called StaTIX, which uses the cosine similarity metric and is based on community detection. Property graph (PG) schema inference in [66] involves inferring types, basic and complex data types, overlapping types, and node hierarchies.

4.3. Discussion: Current Trends and Open Issues

Graph databases are focused on efficiently storing and querying highly connected data. They can use various data models for graphs and their data extensions. Some examples of graph databases include labelled property graphs [69], RDF graphs [58], and hypergraphs [70]. One of the main advantages of using a graph store is that graph stores are designed to handle large amounts of data and provide persistence, ACID properties, CRUD operations, commits, indexing, and other database features. This means running graph algorithms as often as required without recreating the graph each time possible.

The role of GDBMSs in managing text graph databases and facilitating knowledge extraction is explored through the research questions that guide our study.

- RQ_1. What are the capabilities and limitations of graph databases in capturing syntax and semantics within textual content?

Modelling text content with labelled property graphs involves representing the data as a set of nodes and edges with properties. Graph databases represent relationships between entities in a text, which is crucial for capturing both syntax and semantics. Nodes can represent words, phrases, or entire sentences, while edges represent the relationships between these entities (e.g., subject-verb-object structures, dependencies, or co-references). This allows for a more nuanced representation of how different parts of a text are related. However, building a graph that accurately captures both the syntax and semantics of text requires sophisticated parsing and entity-relation extraction techniques. Errors in this process can lead to incorrect or incomplete representations, which can skew the analysis and insights derived from the graph.

- RQ_2. How can a text graph database address missing information, such as incomplete, low-quality, or contradictory annotations, and what are the implications for querying and maintenance?

Graph databases are flexible, allowing for the dynamic addition of new nodes and edges as more text is processed. This is particularly useful for evolving language models where new words, meanings, or relationships can be added without requiring a complete overhaul of the existing structure. Graphs can easily integrate external knowledge sources to enrich the semantic understanding of textual content. This integration can help in disambiguating terms and providing a deeper context for interpreting text. While graphs can represent multiple meanings or interpretations of words, handling ambiguity and polysemy accurately requires complex disambiguation processes. Without proper context or additional disambiguation mechanisms, the graph may fail to capture the true meaning of a text.

- RQ_3. How do graph databases enable the extraction of both explicit and implicit knowledge?

Graph databases support powerful inference capabilities, enabling the extraction of implicit knowledge from explicit relationships.

- Explicit Knowledge: Direct queries on the graph can extract explicit knowledge, such as relationships between entities, specific properties of nodes, or predefined paths. For example, in a graph where nodes represent medical conditions and treatments, a query can directly retrieve all treatments associated with a specific condition. Pattern matching queries allow for the extraction of known relationships and connections within the graph (e.g., "Find all patients who have been treated for hypertension with beta-blockers").
- Implicit Knowledge: Graph traversal queries enable the exploration of relationships that are not immediately obvious. By traversing paths between nodes, these databases can uncover hidden connections and dependencies, revealing implicit knowledge. For instance, traversing a medical knowledge graph might reveal an indirect link between two seemingly unrelated symptoms through a common underlying condition.

- RQ_4. What querying possibilities are enabled by text graph databases?

The challenges and barriers associated with textual content processing with graph databases include scalability, partitioning, processing complexity, and hardware configurations. Managing and querying large graphs can be computationally intensive, leading to performance bottlenecks, especially when dealing with real-time data or large-scale text corpora. For example, scaling graph data by distributing it in a network is much more complex than scaling simpler data models and is still a work in progress [71]. Mining complete frequent patterns from graph databases is also challenging since supporting operations are computationally costly. Also, partitioning and processing graph-structured data in parallel and distributed systems is difficult.

- Graph analytics queries are implemented by GDBMSs with built-in graph algorithms such as PageRank, community detection, and similarity scoring. These algorithms help uncover implicit structures and patterns within the graph, such as identifying influential nodes, clustering similar entities, or finding potential correlations that are not directly stated in the textual content.
- Rule-based inference queries allow to infer new knowledge from existing data based on predefined rules. This allows the database to deduce new relationships or classify entities in ways that were not explicitly stated in the text.

5. Artificial Intelligence for Analysing and Discovering Knowledge from Text Graphs

Extracting information from text is the first step to transform that information into knowledge and valuable insights. These insights may take the form of discoveries or the confirmation and verification of previous hypotheses by creating new connections within our existing understanding of a particular domain. Artificial intelligence (AI) models including machine learning models and neural networks are used for: processing structured textual content with models that capture knowledge within documents. Models are then used for inferring, discovering and classifying new content. The application of AI models raw or structured text must be done through pipelines that prepare the data, train and calibrate the models. These models capture (implicitly or explicitly) the knowledge contained in texts.

5.1. Data Science Pipelines for Modelling Text Content

A data science pipeline involves a series of steps to gather raw data from multiple sources, analyse it, and present the results in an understandable format. The general artificial intelligence pipeline consists of four stages [72]: data management, model learning, model verification, and model deployment. The first three stages involve the activities required to produce machine learned models, and together, they are referred to as the machine learning workflow. The fourth stage, model deployment, involves the integration of machine learning models into an operational system alongside components developed using traditional software and system engineering methods.

A pipeline can be designed to constructing, exploring, and analysing text graphs. General templates for these tasks can be adapted based on the algorithms and strategies used. For example, a graph construction pipeline from textual input might begin by employing information retrieval strategies to create a raw graph where relevant terms from the document are represented as nodes. Contextual rules can be established to connect these nodes, such as linking terms that represent characters if they appear within five words of each other in a sentence. The resulting raw graph can then be refined by pruning, removing redundant, isolated, or irrelevant nodes to create a more meaningful and focused representation.

5.1.1. Building a Data Science Pipeline for Representing Textual Content

Building a graph representing textual content from a document dataset using machine learning models involves several stages, each focused on processing the text, extracting meaningful entities and relationships, and constructing the graph. Figure 3 shows the pipeline implementing the construction of a graph using the example of medical case texts to illustrate the extraction and analysis of meaningful relationships between medical entities.

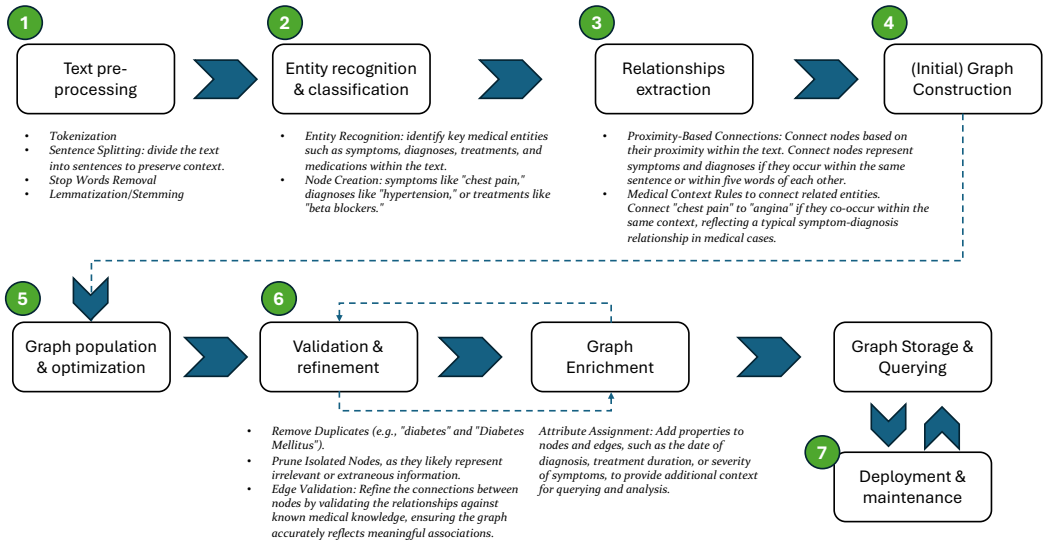

Figure 3. Pipeline for building a text graph.

[1] Text preprocessing. The process begins with document collection, where a diverse dataset is gathered, including text from sources such as articles, clinical reports, emails, or any other relevant textual data. Once collected, the text undergoes preprocessing to prepare it for analysis. This involves tokenization, which breaks the text down into individual words or phrases, known as tokens. Lemmatization or stemming is then applied to normalize these tokens to their root form, ensuring consistency across the dataset. Common words that do not add significant meaning, such as "the" or "and", are removed during the stop words removal step. Finally, the text is split into individual sentences, making it easier to analyse and extract meaningful information in subsequent steps.

[2] Entity Recognition and Classification. Named entity recognition (NER) involves using a pre-trained model or training a custom model to identify and classify key entities within the text, such as persons, organizations, locations, dates, or medical terms. Once these entities are recognized, entity linking is performed, where each entity is connected to a corresponding entry in a knowledge base or ontology—such as linking "New York" to the concept of a city in a geographical database. After linking, the entities are classified into predefined categories based on their roles within the text, such as "Person", "Location", "Disease", or "Medication", helping to organise and structure the extracted information for further analysis.

[3] Relationship Extraction. Dependency parsing involves using models to analyse the syntactical structure of sentences, identifying relationships between words such as subject-verb-object connections. This process helps to map out how different words interact within a sentence, providing a foundation for understanding the text's meaning. Once these relationships are identified, machine learning models are applied to classify them, determining specific types of connections between entities, such as "treated with", "located in", or "works for". Additionally, co-reference resolution is performed to ensure that references to the same entity—such as "John" and "he"—are correctly linked, maintaining consistency in the extracted relationships throughout the text.

[4] Initial Graph construction. In the graph construction phase, nodes are created for each identified entity, with each node labelled according to the entity type, such as "Person", "Location", or "Condition". These nodes are then enriched with properties that reflect attributes extracted from the text, like age, date, or description, providing additional context for each entity. Following node creation, edges are established between nodes to represent the relationships identified during the text analysis. Each edge is labelled to indicate the type of relationship, such as "diagnosed_with" or "treated_with", and, where applicable, properties like the date or context of the relationship are assigned to these edges, further enhancing the graph's ability to capture and represent the intricacies of the textual content.

[5] Graph population and optimization. In the graph population and optimization stage, the constructed nodes and edges are inserted into a GDMBS, creating a structured representation of the textual content. To ensure efficient access and analysis, indexing is applied to frequently queried nodes and edges, enhancing the speed and performance of graph traversal and querying. Additionally, the graph structure is optimised by merging similar nodes or relationships, which reduces redundancy and streamlines the overall graph, making it more efficient for storage, retrieval, and further analysis.

[6] Validation and refinement. In the validation and refinement phase, the accuracy of the graph structure is validated by comparing it against a ground truth or leveraging feedback from domain experts to ensure that the relationships and entities accurately represent the underlying data. Based on this validation, the machine learning models used for entity recognition, relationship extraction, and graph-based predictions are refined to address any discrepancies or inaccuracies identified. This process of refinement enhances the precision and reliability of the graph. Additionally, continuous learning is implemented, where models are regularly updated and retrained as new documents are added to the dataset. This ongoing process ensures that the graph remains accurate, relevant, and reflective of the most current data available.

[7] Deployment and maintenance. In the deployment and maintenance phase, the graph database is deployed for real-time applications such as question answering systems, recommendation engines, or decision support systems, where it can be actively used to derive insights and support decision-making. Maintenance involves regularly updating the graph with new data to ensure it remains current and relevant. Additionally, machine learning models are retrained as needed to accommodate evolving data patterns and improve system accuracy. Continuous monitoring of the graph's performance and accuracy is essential to ensure the system functions optimally and delivers reliable results over time.

5.1.2. Data Science as First-Class Citizens

Treating the pipeline as a first-class citizen within a platform offers significant benefits in terms of visibility, traceability, and overall operational efficiency. By allowing for clear workflow visualization, stakeholders can easily see the entire process, understand each step, and trace the flow of data through the pipeline, which enhances collaboration across teams. This visibility ensures that all members are on the same page, facilitating discussions around optimizations and improvements. From an operational perspective, having the

pipeline as a core component enables automated management of tasks like scheduling, monitoring, and error handling, thereby reducing manual intervention and increasing efficiency. Proactive maintenance is supported by built-in monitoring and logging, allowing for the early detection and resolution of issues, which helps the pipeline run smoothly and reliably.

Moreover, treating the pipeline as a first-class citizen promotes reusability and scalability. Modular components can be reused across different projects, ensuring consistency and reducing development time. Platforms like Airflow or Kubeflow provide scalable infrastructure, supporting dynamic expansion as data volumes grow or new tasks are added, ensuring that the system can handle increasing demands without degradation in performance. The pipeline's visibility also supports continuous improvement, as teams can iteratively enhance the workflow, integrate new technologies, and optimise processes based on real-time feedback. Adaptive learning through retrainable machine learning models ensures that the system remains accurate and relevant as data patterns evolve.

Finally, a well-documented and visible pipeline facilitates compliance and auditing, making it easier to demonstrate adherence to regulatory requirements related to data processing and security. The structured approach to handling data within the pipeline ensures that governance policies are consistently applied, reducing the risk of data mishandling or breaches, and ensuring that the organization meets its data governance standards.

5.1.3. Implementing Data Science Pipelines

A pipeline for building and maintaining a graph database from textual content can be designed as a workflow within existing data processing and machine learning platforms such as Apache Airflow, Kubeflow, or Azure Data Factory.

A well-designed pipeline benefits from modular design, where the workflow is broken down into discrete tasks such as data collection, preprocessing, entity recognition, relationship extraction, graph construction, and validation. This task segmentation allows for efficient dependency management, ensuring that each step is executed in the correct order (e.g., preprocessing before entity recognition). Additionally, by leveraging parallel processing, the pipeline can run independent tasks simultaneously, optimizing both speed and resource efficiency.

Integration with existing tools is crucial, enabling seamless data collection from various sources (e.g., cloud storage, databases), utilizing machine learning models for tasks like named entity recognition (NER) and dependency parsing, and connecting directly to a GDBMS for smooth data insertion and querying. Automation and scheduling enhance the pipeline's efficiency by allowing automated execution at regular intervals and event-driven triggers that activate the pipeline in response to specific events, such as the arrival of new data. Real-time monitoring tools are employed to track task execution and identify bottlenecks or failures, while detailed logging supports troubleshooting and performance analysis. The pipeline is designed to be scalable and flexible, with resources that can dynamically adjust based on data volume and task complexity, and the ability to easily modify the workflow to accommodate new tasks or changes in existing processes, ensuring continuous adaptation to evolving requirements.

5.2. Vectorial Representation of Text Graphs

Vectorial representations or embeddings are powerful tools for analysing and understanding complex graph data. An essential challenge is a decision on the embedding dimensionality. Longer embeddings preserve more information and induce higher time and space complexity than sorter embeddings. Users must make a trade-off based on the requirements (https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007, accessed on 19 August 2024).

Vectorial representations or embeddings capture the graph topology, vertex-to-vertex relationship, and other relevant information about graphs, subgraphs, and vertices. These representations can encode each vertex with its vector representation to perform visualisa-

tion or prediction on the vertex level. For example, vertices can be visualised in a 2D plane, or new connections can be predicted based on vertex similarities.

Building a vectorial (vector-based) representation of a text graph involves transforming the graph structure, which inherently captures relationships and entities, into a format that can be processed by machine learning models, particularly those that require fixed-dimensional input.

Graph Neural Networks for Building Text Graphs

Graph neural networks (GNNs) are a type of neural network designed to work with graph data structures. GNNs are used to predict nodes, edges, and graph-based tasks. The key design element of GNNs is pairwise message passing, such that graph nodes iteratively update their representations by exchanging information with their neighbours. This method allows dealing with dynamic graphs [73]. GNNs can also allow dealing with heterogeneous data represented as heterogeneous graphs [74]. The principle involves creating a view from a heterogeneous graph that contains nodes and edges of the same type, and then generating a vectorial representation for each node type, as illustrated in Figure 4. A heterogeneous graph is characterized by multiple types of objects and links, making it suitable for modeling complex systems and relational data. In the figure, nodes of different colors represent different types of nodes within the graph (e.g., d and e, b and c, g and f in the Figure are nodes of the same type). The objective is to analyze and represent these heterogeneous components in a unified manner by transforming them into their corresponding vectorial forms.

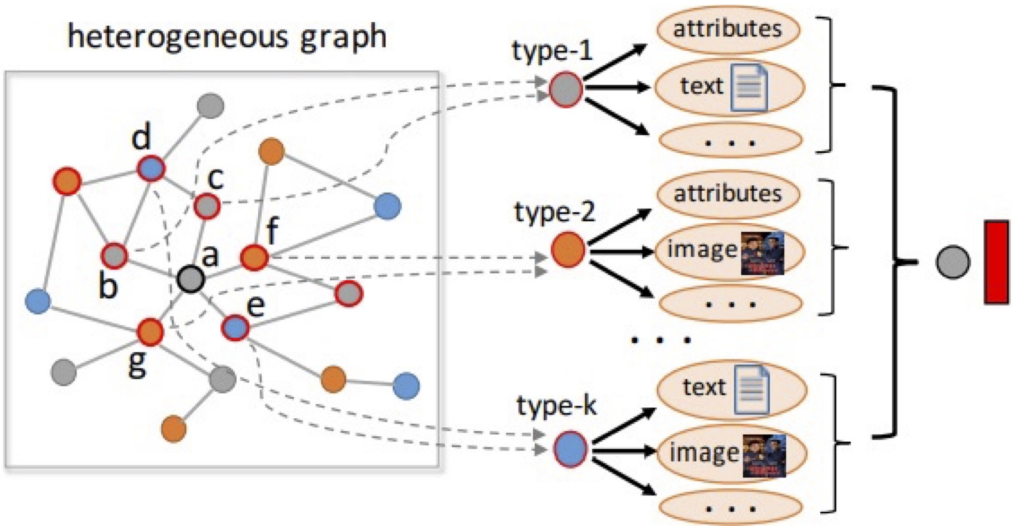

Figure 4. Heterogeneous graph.

Graph embeddings are the transformation of property graphs to a vector or a set of vectors. These embeddings can be used to make predictions on the graph level and to compare or visualise whole graphs. For instance, graph embeddings can be used to compare chemical structures. For example, suppose we have two graphs representing social networks, and we want to compare the similarity of their community structures. We can use a graph embedding algorithm to generate embeddings for each graph node. Then, we can use a similarity measure, such as cosine similarity, to compare the embeddings of corresponding nodes in the two graphs. This will give us a measure of how similar the community structures of the two graphs are.

Relevant application domains for GNNs include natural language processing, social networks, citation networks, molecular biology, chemistry, physics and NP-hard combinatorial optimisation problems. Several open-source libraries implementing graph neural

networks are available, such as PyTorch Geometric (PyTorch), TensorFlow GNN (TensorFlow), and jraph (Google JAX).

To preprocess the text graph and extract knowledge from texts, the pipelines should include tasks for transforming the graphs into vectorial representations that can be used in training, testing, and evaluating GNN models. Depending on the type of graph (i.e., heterogeneous or homogeneous graph, directed or undirected, properties in the nodes and/or edges), the preprocessing pipeline can change. For example, nodes and edges can be extracted separately from their properties and transformed into vectors using text2vec. In contrast, selecting graph views from a heterogeneous graph, such that the nodes and edges are the same type, and then computing a global vector representing the whole graph. Calibrating the transformation models is a recurrent process that data science pipelines can exhibit (see Figure 5, https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007, accessed on the 25 September 2024).

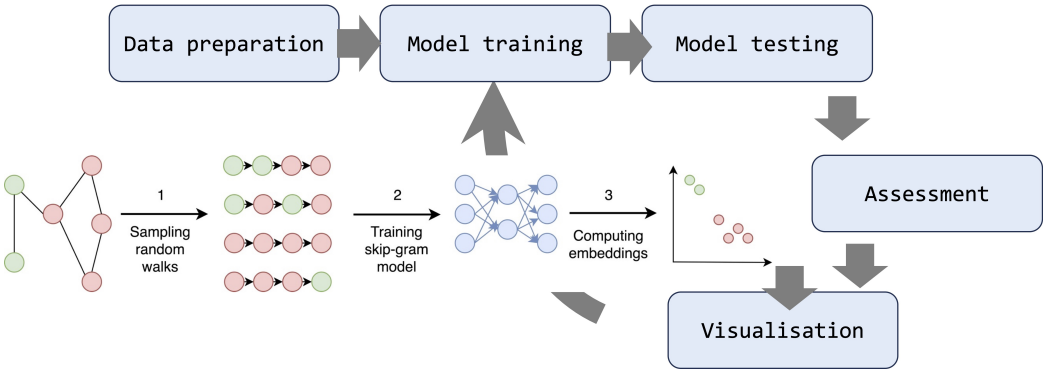

Figure 5. Graph processing pipeline example.

In the figure (left-hand side at the bottom), we start with a graph containing different types of nodes, represented in red and blue. A sampling process, using random walks, leads to several paths that traverse both red and blue nodes. In a random walk, given a graph and a starting node, we randomly select a neighboring node and move to it. This process is repeated by selecting random neighbors at each step, generating a sequence of points, which constitutes the random walk on the graph. These random walks are then used to train a skip-gram model, resulting in a graph that connects the nodes visited during the walks.

To clarify, the skip-gram model is a technique used to create word embeddings by predicting the surrounding words based on a target word. For example, when reading a sentence, the skip-gram model allows one to infer the words that typically appear before or after a specific word. By applying this technique to the graph, embeddings are generated, which allow the grouping of similar nodes (red and blue) that frequently occur together, much like how words that commonly appear together in a sentence are clustered in word embeddings.

5.3. Discussion: Current Trends and Open Issues

Processing natural language involves enabling programs to produce relevant documents in good shape and automatically marking and extracting relevant information hidden in the text. Preparing textual data and moving it to a target system for analytics (textual extraction transformation and loading—ETL) involves identifying patterns and trends in the data. Standard natural language processing approaches gain valuable knowledge in business process complexity, but this is still an active area of research. An example of a challenging application is processing the textual content in clinical cases [75]. Identifying important clinical terms stated in digital text, such as treatments, disorders, and symptoms, can be challenging; locating key entities within the text or in clinical text may be in an unstructured format, making it difficult to process; and repeated words make it difficult to extract meaningful information.

Machine learning can be computationally expensive, and using models from related domains can help reduce the cost of training new models. Transfer learning [76] is a technique that allows a model trained in one domain to be used in another domain as long as the domains are similar enough that the features learned in the source domain apply to the target domain. When this is the case, all or part of a model can be transferred to reduce the cost of training. Convolutional neural networks (CNNs) are particularly well-suited for partial model transfer, as their convolutional layers encode features in the input space. In contrast, their fully connected layers encode reasoning based on those features. This allows for transferring learned features from one domain to another, reducing the computational cost of training new models.

The following discussion addresses current trends and open issues in the role of artificial intelligence for extracting information and knowledge from textual content, guided by the research questions of this study.

- RQ_1. What are the capabilities and limitations of vectorial representations in capturing syntax and semantics within textual content?

Neural networks (NNs), offer powerful tools for modeling textual content and extracting knowledge. These models convert text into high-dimensional vectors that capture semantic meaning and contextual relationships within the text. When applied within GNNs, these vectorial representations enable the network to understand and propagate information across nodes and edges in a graph, capturing complex dependencies and relationships inherent in the text. This allows for more nuanced and comprehensive knowledge extraction, as GNNs can leverage the structural information encoded in graphs to improve tasks like entity recognition, sentiment analysis, and relationship extraction. Similarly, traditional neural networks can use these vector embeddings to perform tasks such as text classification, summarization, and translation, where the model learns to generate meaningful outputs based on the encoded textual features. By combining these vector-based AI models with graph-based approaches, we can enhance the ability to model textual content more effectively, enabling deeper insights and more accurate knowledge extraction from complex datasets.

- RQ_{2a}. How can machine learning models address missing information, such as incomplete, low-quality, or contradictory annotations?

Dealing with missing information can be done using gradual pattern extraction techniques [77] and building an information extraction pipeline (https://neo4j.com/blog/text-to-knowledge-graph-information-extraction-pipeline/, accessed on 19 August 2024). The pipeline involves taking unstructured text inputs, processing them with natural language processing (NLP) techniques, and using the resulting structures to populate or enrich a knowledge graph. This can help to fill in missing values and improve the overall quality of the data.

The approaches to making annotations evolve according to the evolution of repositories, including using an information extraction pipeline or gradual pattern extraction techniques [48]. In the first case, unstructured text inputs are processed with natural language processing (NLP) techniques, and the resulting structures populate or enrich a knowledge graph. In the second case, extracting gradual patterns from property graphs provides end-users with tools for mining correlations in the data when missing values exist.

- RQ_{2b}. How can a vectorial representations of text graphs address missing information, such as incomplete, low-quality, or contradictory annotations, and what are the implications for querying and maintenance?

Vectorial representations of text graphs provide a robust mechanism for addressing the challenges of missing, low-quality, and contradictory information. They enhance the accuracy and reliability of querying while reducing the maintenance burden, making them an essential tool for extracting meaningful insights from complex and imperfect textual datasets.

- Addressing Missing and Incomplete Information. Vectorial representations of text graphs can effectively address issues of missing information, incomplete or low-quality data, and contradictory annotations by leveraging the inherent ability of vector spaces to capture and generalize patterns across the data.

Vectorial representations, such as word or node embeddings, are generated by learning from the context in which entities and relationships appear. When some information is missing or incomplete, these vectors can help infer the likely characteristics of missing data based on the patterns observed in the rest of the graph. For instance, if a particular symptom is missing in some medical records, the model can estimate its presence or relevance based on the relationships and similarities with other well-annotated cases.

Vectorial embeddings naturally smooth over noisy or sparse data by clustering similar entities together in the vector space. This clustering can reduce the impact of low-quality data by drawing on the collective context of similar, higher-quality data points. This helps maintain the integrity of the graph's representation, even when individual annotations are unreliable.

- Handling Contradictory Annotations. Vectorial models can capture the different contexts in which similar entities appear, allowing the model to distinguish between contradictory annotations based on their respective contexts. For example, in a text graph where "headache" might be annotated as a symptom of both stress and a neurological condition, vector representations can help differentiate these contexts, reducing the ambiguity in querying and analysis.

Vectorial representations often support probabilistic approaches to handling contradictory data. The model can assign probabilities to different annotations based on their consistency with the overall graph structure, allowing for more nuanced querying and decision-making. This means that during querying, the model can weigh different annotations according to their likelihood, leading to more informed and balanced results.

- Implications for Querying. With vectorial representations addressing missing and contradictory information, queries against the graph can return more accurate and contextually relevant results. The model's ability to infer and generalize from incomplete data ensures that queries are less likely to fail or return incorrect results due to gaps or errors in the dataset.

Vectorial representations enable more sophisticated, semantic-level queries that go beyond simple keyword matching. For example, querying for patients with symptoms similar to a rare condition might yield results even when exact matches are missing, thanks to the semantic similarity captured in the vector space.

- Implications for Maintenance. The ability of vectorial models to generalize from incomplete or low-quality data reduces the need for constant manual curation and correction of the dataset. This can significantly lower maintenance overheads, as the system becomes more resilient to imperfections in the data.

Vectorial representations can be continuously updated as new data comes in, allowing the model to adapt to new patterns and correct earlier inaccuracies or gaps. This continuous learning approach ensures that the graph remains relevant and accurate over time, even as the underlying data evolves.

- RQ_3. How do vectorial representations enable the extraction of both explicit and implicit knowledge?

Vectorial representations are powerful tools for extracting both explicit and implicit knowledge because they capture not only the direct, surface-level relationships in data but also the deeper, context-driven connections that reveal underlying patterns and meanings. This dual capability enhances the ability to derive meaningful insights from complex datasets, enabling more informed decision-making and advanced analysis.

[1] Extraction of explicit knowledge. Vectorial representations encode explicit knowledge directly from the data, such as the meaning of words, relationships between entities, and their attributes. For example, in a medical context, a word embedding might directly capture the relationship between terms like "diabetes" and "insulin" based on their co-occurrence in the text. These vectors encode the explicit connections, allowing for straightforward retrieval of information through querying or similarity searches.

Explicit relationships, such as synonyms or related terms, are naturally clustered together in vector space. This clustering makes it easy to extract explicit knowledge by grouping or categorizing similar entities. For instance, all medications related to a specific condition might be clustered together, facilitating quick identification of relevant treatments.

[2] Extraction of implicit knowledge. Vectorial representations capture the broader context in which entities appear, allowing the model to infer relationships and meanings that are not explicitly stated. For instance, if "fatigue" frequently appears in medical cases involving both "anemia" and "thyroid disorders", the model can implicitly associate fatigue with these conditions, even if this connection is not directly annotated.

Implicit knowledge is often derived from the semantic similarity between different entities. Vectors representing similar concepts or entities will be close to each other in the vector space, allowing the model to infer relationships or attributes that are not explicitly connected in the original data. For example, if "chest pain" and "angina" are represented by similar vectors, the model might infer that treatments effective for angina could also be relevant for cases labelled with chest pain.

Vectorial representations enable models to recognize patterns across large datasets, even when these patterns are not explicitly defined. This allows for the extraction of implicit knowledge, such as potential correlations between symptoms and conditions that were not previously identified. For instance, a model might detect that a combination of certain symptoms typically precedes a specific diagnosis, offering predictive insights based on implicit patterns.

- RQ_4. What querying possibilities are enabled by vectorial text graph representations?

The method used to query graphs depends on the type of representation used and the user's specific needs. For instance, vector similarity search is a technique that searches for vectors based on their similarity to a given query (A vector search database, also known as a vector similarity search engine or vector database, is a type of database that is designed to store, retrieve, and search for vectors based on their similarity given a query https://labelbox.com/blog/how-vector-similarity-search-works/, accessed on 19 August 2024). This approach is commonly used in applications such as image retrieval, natural language processing, and recommendation systems.

By mapping entities into a continuous vector space, vectorial representations allow for the discovery of latent relationships—those that are not directly observable in the raw data. Queries can leverage these latent relationships to extract implicit knowledge, providing deeper insights that go beyond surface-level connections.

Vectorial representations enable flexible querying, where the model can return relevant results even for incomplete or vague queries by leveraging the semantic proximity of vectors. For example, a query about "treatments for severe headaches" might return results related to migraines, even if "migraines" were not explicitly mentioned, because the vectors for "severe headaches" and "migraines" are close in the vector space.

6. Conclusions and Outlook

Graph technology underpins modern data and analytics, offering powerful capabilities that enhance user collaboration, machine learning models, and the development of explainable AI. Diverse objectives drive the disciplines involved in text processing, each tailored to specific analytical needs and applications.

1. Content extraction: This involves using linguistic techniques to delve into the language and extract meaningful content.
2. Knowledge representation: This assumes that textual content defines a network of representative concepts, semantic relations, and associated consistency rules that represent the knowledge contained in the text, including knowledge that can be inferred from it.
3. Efficient query execution: This involves selecting appropriate graph data models to represent textual content so that specific queries can be answered efficiently.
4. Knowledge modelling, discovery, and prediction: Given textual content represented as graphs, this involves modelling how concepts weave content by connecting words, sentences, and groups of sentences with the hypothesis that new knowledge can be produced and predicted. For example, graph machine learning, or geometric machine learning, can learn from complex data such as graphs and multi-dimensional points. Its applications have been relevant in fields such as biochemistry, drug design, and structural biology (https://www.gartner.com/smarterwithgartner/gartner-top-10-data-and-analytics-trends-for-2021, accessed on 19 August 2024).

The aim of textual data analysis extends beyond mere extraction and inference of knowledge through artificial intelligence techniques like machine learning, knowledge representation, and reasoning. It also involves generating new insights from textual content. This multifaceted challenge calls for a unified integration of results to preserve and juxtapose the knowledge derived from texts, typically represented in graphical formats within various frameworks of consistency. Efforts have already been made to dynamically model textual content using ontologies and graph databases, focusing on its evolution and the methodologies for validation within ongoing learning processes. With the rise of advanced technologies such as large language models, generative AI, and reinforcement learning, the field is moving towards more sophisticated data management paradigms. These paradigms need to adapt to the specific encoding requirements of data, like vectors and matrices, treating these structures as integral elements for optimised processing. Emerging vectorial and neural data management systems are thus designed to ensure the persistence, indexing, and maintenance necessary to support scalable learning models efficiently.

Our current and future work tackles the problem from a multidisciplinary perspective, focusing on two key research lines. The first involves developing an analytics query model based on the concept of data science pipelines for modelling textual content. Given the intricacies of language, these pipelines must enable the seamless integration of results from various disciplines into complex analytical queries, with human guidance in their design and execution. The complexity of natural language also necessitates domain-specific strategies. Our work in Biodiversity and Gender Studies, for example, is advancing data science pipelines that are not only tailored to these domains but also allow human intervention to control and calibrate their execution. Additionally, we are exploring conversational approaches to designing these pipelines, ensuring they are user-friendly and adaptable. The second research line focuses on exploring vectorial database management systems, particularly their capacity to execute data science pipelines while minimizing impedance mismatch between vectorial representations and the data management and query evaluation processes of structured data models. This exploration aims to enhance the efficiency and accuracy of textual content analysis in these complex environments.

Funding: This research was partially funded by the GDR MADICS of the French Council of Scientific Research (CNRS).

Data Availability Statement: Data are contained within the article.

Acknowledgments: The paper is inspired by the discussion in the national actions DOING/ROCED panel in the national symposium MADICS 2022 in Lyon. I thank the panellists who shared their insight and perspectives about the questions addressed in this paper: Donatello Conte (Polytech Tours, LIFAT), Nathalie Hernandez (Université de Toulouse, IRIT), Catherine Roussey (INRAE), Agata Savary (Université Paris-Saclay, LISN), Nicolas Travers (ELSIV, Centre de Recherche Da Vinci).

It also follows up the insights generated by the work done in the coordination action DOING, funded by the CNRS-MADICS program.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Turgunova, N.; Turgunov, B.; Umaraliyev, J. Automatic text analysis. syntax and semantic analysis. In *Engineering Problems and Innovations*; TATUFF-EPAI: Chinobod, Uzbekistan, 2023.
2. Nadkarni, P.M.; Ohno-Machado, L.; Chapman, W.W. Natural language processing: An introduction. *J. Am. Med. Inform. Assoc.* **2011**, *18*, 544–551. [CrossRef] [PubMed]
3. Idnay, B.; Dreisbach, C.; Weng, C.; Schnall, R. A systematic review on natural language processing systems for eligibility prescreening in clinical research. *J. Am. Med. Inform. Assoc.* **2022**, *29*, 197–206. [CrossRef] [PubMed]
4. Fanni, S.C.; Febi, M.; Aghakhanyan, G.; Neri, E. Natural language processing. In *Introduction to Artificial Intelligence*; Springer: Berlin, Germany, 2023; pp. 87–99.
5. Trivedi, A.; Pant, N.; Shah, P.; Sonik, S.; Agrawal, S. Speech to text and text to speech recognition systems—A review. *IOSR J. Comput. Eng* **2018**, *20*, 36–43.
6. Luerkens, D.W.; Beddow, J.K.; Vetter, A.F. Theory of morphological analysis. In *Particle Characterization in Technology*; CRC Press: Boca Raton, FL, USA, 2018; pp. 3–14.
7. Chomsky, N. Systems of syntactic analysis. *J. Symb. Log.* **1953**, *18*, 242–256. [CrossRef]
8. Chowdhary, K.; Chowdhary, K. Natural language processing. In *Fundamentals of Artificial Intelligence*; Springer: New Delhi, India, 2020; pp. 603–649.
9. Eisenstein, J. *Introduction to Natural Language Processing*; MIT Press: Cambridge, MA, USA, 2019.
10. Maulud, D.H.; Zeebaree, S.R.; Jacksi, K.; Sadeeq, M.A.M.; Sharif, K.H. State of art for semantic analysis of natural language processing. *Qubahan Acad. J.* **2021**, *1*, 21–28. [CrossRef]
11. Geeraerts, D. *Theories of Lexical Semantics*; OUP Oxford: Oxford, UK, 2009.
12. Wang, C.; Zhou, X.; Pan, S.; Dong, L.; Song, Z.; Sha, Y. Exploring relational semantics for inductive knowledge graph completion. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 28 February–1 March 2022; Volume 36, pp. 4184–4192.
13. Potter, J. Discourse analysis. In *Handbook of Data Analysis*; Sage: London, UK, 2004; pp. 607–624.
14. Chauhan, K.; Jain, K.; Ranu, S.; Bedathur, S.; Bagchi, A. Answering Regular Path Queries through Exemplars. *Proc. VLDB Endow.* **2021**, *15*, 299–311. [CrossRef]
15. Arul, S.M.; Senthil, G.; Jayasudha, S.; Alkhayyat, A.; Azam, K.; Elangovan, R. Graph Theory and Algorithms for Network Analysis. *E3S Web Conf. EDP Sci.* **2023**, *399*, 08002. [CrossRef]
16. Zhang, P.; Wang, T.; Yan, J. PageRank centrality and algorithms for weighted, directed networks. *Phys. A Stat. Mech. Appl.* **2022**, *586*, 126438. [CrossRef]
17. Garrido-Muñoz, I.; Montejo-Ráez, A.; Martínez-Santiago, F.; Ureña-López, L.A. A survey on bias in deep NLP. *Appl. Sci.* **2021**, *11*, 3184. [CrossRef]
18. Dev, S.; Sheng, E.; Zhao, J.; Amstutz, A.; Sun, J.; Hou, Y.; Sanseverino, M.; Kim, J.; Nishi, A.; Peng, N.; et al. On measures of biases and harms in NLP. *arXiv* **2021**, arXiv:2108.03362.
19. Hutto, C.; Gilbert, E. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In Proceedings of the International AAAI Conference on Web and Social Media, Ann Arbor, MI USA, 1–4 June 2014; Volume 8, pp. 216–225.
20. Loper, E.; Bird, S. Nltk: The natural language toolkit. *arXiv* **2002**, arXiv:cs/0205028.
21. Bolukbasi, T.; Chang, K.W.; Zou, J.Y.; Saligrama, V.; Kalai, A.T. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In Proceedings of the Advances in Neural Information Processing Systems 29, Barcelona, Spain, 5–10 December 2016.
22. Zhang, Y.; Ramesh, A. Learning fairness-aware relational structures. In *ECAI 2020*; IOS Press: Tepper Drive Clifton, VA, USA, 2020; pp. 2543–2550.
23. Wiegreffe, S.; Pinter, Y. Attention is not not explanation. *arXiv* **2019**, arXiv:1908.04626.
24. Hardt, M.; Price, E.; Srebro, N. Equality of opportunity in supervised learning. In Proceedings of the Advances in Neural Information Processing Systems 29, Barcelona, Spain, 5–10 December 2016.
25. Raji, I.D.; Smart, A.; White, R.N.; Mitchell, M.; Gebru, T.; Hutchinson, B.; Smith-Loud, J.; Theron, D.; Barnes, P. Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, Barcelona Spain, 27–30 January 2020; pp. 33–44.
26. Jobin, A.; Ienca, M.; Vayena, E. The global landscape of AI ethics guidelines. *Nat. Mach. Intell.* **2019**, *1*, 389–399. [CrossRef]
27. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the Advances in Neural Information Processing Systems 26, Lake Tahoe, NV, USA, 5–10 December 2013.
28. Le, Q.; Mikolov, T. Distributed representations of sentences and documents. In Proceedings of the International Conference on Machine Learning. PMLR, Beijing, China, 22–24 June 2014; pp. 1188–1196.

29. Ma, S.; Sun, X.; Li, W.; Li, S.; Li, W.; Ren, X. Query and output: Generating words by querying distributed word representations for paraphrase generation. *arXiv* **2018**, arXiv:1803.01465.
30. Kaddari, Z.; Mellah, Y.; Berrich, J.; Belkasmi, M.G.; Bouchentouf, T. Natural language processing: challenges and future directions. In Proceedings of the International Conference on Artificial Intelligence & Industrial Applications, Meknes, Morocco, 19–20 March 2020; Springer: Cham, Switzerland, 2020; pp. 236–246.
31. Khurana, D.; Koli, A.; Khatter, K.; Singh, S. Natural language processing: State of the art, current trends and challenges. *Multimed. Tools Appl.* **2023**, *82*, 3713–3744. [CrossRef]
32. Savary, A.; Silvanovich, A.; Minard, A.L.; Hiot, N.; Ferrari2D, M.H. Relation Extraction from Clinical Cases. In Proceedings of the New Trends in Database and Information Systems: ADBIS 2022 Short Papers, Doctoral Consortium and Workshops: DOING, K-GALS, MADEISD, MegaData, SWODCH, Turin, Italy, 5–8 September 2022; Proceedings; Springer Nature: Berlin, Germany, 2022; p. 353.
33. Carriere, J.; Shafi, H.; Brehon, K.; Pohar Manhas, K.; Churchill, K.; Ho, C.; Tavakoli, M. Case report: Utilizing AI and NLP to assist with healthcare and rehabilitation during the COVID-19 pandemic. *Front. Artif. Intell.* **2021**, *4*, 613637. [CrossRef]
34. Jozefowicz, R.; Vinyals, O.; Schuster, M.; Shazeer, N.; Wu, Y. Exploring the limits of language modeling. *arXiv* **2016**, arXiv:1602.02410.
35. Kouadri, W.M.; Ouziri, M.; Benbernou, S.; Echihabi, K.; Palpanas, T.; Amor, I.B. Quality of sentiment analysis tools: The reasons of inconsistency. *Proc. VLDB Endow.* **2020**, *14*, 668–681. [CrossRef]
36. Rossiello, G.; Chowdhury, M.F.M.; Mihindukulasooriya, N.; Cornec, O.; Gliozzo, A.M. Knowgl: Knowledge generation and linking from text. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 20–27 February 2023; Volume 37, pp. 16476–16478.
37. Chimalakonda, S.; Nori, K.V. An ontology based modeling framework for design of educational technologies. *Smart Learn. Environ.* **2020**, *7*, 1–24. [CrossRef]
38. Al-Aswadi, F.N.; Chan, H.Y.; Gan, K.H. Automatic ontology construction from text: A review from shallow to deep learning trend. *Artif. Intell. Rev.* **2020**, *53*, 3901–3928. [CrossRef]
39. Bienvenu, M.; Leclère, M.; Mugnier, M.L.; Rousset, M.C. Reasoning with ontologies. In *A Guided Tour of Artificial Intelligence Research: Volume I: Knowledge Representation, Reasoning and Learning*; Springer: Cham, Switzerland, 2020; pp. 185–215.
40. Zaihrayeu, I.; Sun, L.; Giunchiglia, F.; Pan, W.; Ju, Q.; Chi, M.; Huang, X. From web directories to ontologies: Natural language processing challenges. In Proceedings of the International Semantic Web Conference, Busan, Republic of Korea, 11–15 November 2007; Springer: Berlin, Germany, 2007; pp. 623–636.
41. Maynard, D.; Bontcheva, K.; Augenstein, I. *Natural Language Processing for the Semantic Web*; Springer: Cham, Switzerland, 2017.
42. Asim, M.N.; Wasim, M.; Khan, M.U.G.; Mahmood, W.; Abbasi, H.M. A survey of ontology learning techniques and applications. *Database* **2018**, *2018*, bay101. [CrossRef] [PubMed]
43. Benbernou, S.; Ouziri, M. Enhancing data quality by cleaning inconsistent big RDF data. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; IEEE: New York, NY, USA, 2017; pp. 74–79.
44. Mikroyannidi, E.; Quesada-Martínez, M.; Tsarkov, D.; Fernández Breis, J.T.; Stevens, R.; Palmisano, I. A quality assurance workflow for ontologies based on semantic regularities. In Proceedings of the Knowledge Engineering and Knowledge Management: 19th International Conference, EKAW 2014, Linköping, Sweden, 24–28 November 2014; Proceedings 19; Springer: Berlin, Germany, 2014; pp. 288–303.
45. Wilson, R.S.I.; Goonetillake, J.S.; Ginige, A.; Indika, W.A. Ontology quality evaluation methodology. In Proceedings of the International Conference on Computational Science and Its Applications, Athens, Greece, 3–6 July 2022; Springer: Berlin, Germany, 2022; pp. 509–528.
46. Sheveleva, T.; Herrmann, K.; Wawer, M.L.; Kahra, C.; Nürnberger, F.; Koepler, O.; Mozgova, I.; Lachmayer, R.; Auer, S. Ontology-Based Documentation of Quality Assurance Measures Using the Example of a Visual Inspection. In Proceedings of the International Conference on System-Integrated Intelligence, Genova, Italy, 7–9 September 2022; Springer: Berlin, Germany, 2022; pp. 415–424.
47. Schneider, T.; Šimkus, M. Ontologies and data management: a brief survey. *KI-Künstl. Intell.* **2020**, *34*, 329–353. [CrossRef] [PubMed]
48. Cardoso, S.D.; Pruski, C.; Da Silveira, M.; Lin, Y.C.; Groß, A.; Rahm, E.; Reynaud-Delaître, C. Leveraging the impact of ontology evolution on semantic annotations. In Proceedings of the Knowledge Engineering and Knowledge Management: 20th International Conference, EKAW 2016, Bologna, Italy, 19–23 November 2016; Proceedings 20; Springer: Berlin, Germany, 2016; pp. 68–82.
49. Pietranik, M.; Kozierkiewicz, A. Methods of managing the evolution of ontologies and their alignments. *Appl. Intell.* **2023**, *53*, 20382–20401. [CrossRef]
50. Ziebelin, M.D.; Pernelle, M.N.; Broisin, M.J.; DesprÈs, M.S.; Rousset, M.M.C.; Jouanot, M.F.; Druette, M.L. Interactive Ontology Modeling and Updating: Application to Simulation-based Training in Medicine. In Proceedings of the 2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Bayonne, France, 27–29 October 2021; pp. 177–182.
51. Espinoza, A.; Del-Moral, E.; Martínez-Martínez, A.; Alí, N. A validation & verification driven ontology: An iterative process. *Appl. Ontol.* **2021**, *16*, 297–337.

52. Ngom, A.N.; Diallo, P.F.; Kamara-Sangaré, F.; Lo, M. A method to validate the insertion of a new concept in an ontology. In Proceedings of the 2016 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), Naples, Italy, 28 November–1 December 2016; IEEE: New York, NY, USA, 2016; pp. 275–281.
53. Tartir, S.; Arpinar, I.B.; Sheth, A.P. Ontological evaluation and validation. In *Theory and Applications of Ontology: Computer Applications*; Springer: Dordrecht, The Netherland, 2010; pp. 115–130.
54. Della Valle, E.; Ceri, S. Querying the semantic web: SPARQL. In *Handbook of Semantic Web Technologies*; Springer: Berlin, Germany, 2011.
55. Hogan, A.; Reutter, J.L.; Soto, A. In-database graph analytics with recursive SPARQL. In Proceedings of the International Semantic Web Conference, Athens, Greece, 2–6 November 2020; Springer: Berlin, Germany, 2020; pp. 511–528.
56. Hogan, A.; Reutter, J.; Soto, A. Recursive SPARQL for Graph Analytics. *arXiv* **2020**, arXiv:2004.01816.
57. Mosser, M.; Pieressa, F.; Reutter, J.; Soto, A.; Vrgoč, D. Querying apis with SPARQL: language and worst-case optimal algorithms. In Proceedings of the Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Greece, 3–7 June 2018; Proceedings 15; Springer: Berlin, Germany, 2018; pp. 639–654.
58. Ali, W.; Saleem, M.; Yao, B.; Hogan, A.; Ngomo, A.C.N. A survey of RDF stores & SPARQL engines for querying knowledge graphs. *VLDB J.* **2022**, *31*, 1–26.
59. Prevoteau, H.; Djebali, S.; Laiping, Z.; Travers, N. Propagation measure on circulation graphs for tourism behavior analysis. In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, 25–29 April 2022; pp. 556–563.
60. Getoor, L.; Machanavajjhala, A. Entity resolution: theory, practice & open challenges. *Proc. VLDB Endow.* **2012**, *5*, 2018–2019.
61. Christophides, V.; Efthymiou, V.; Palpanas, T.; Papadakis, G.; Stefanidis, K. An overview of end-to-end entity resolution for big data. *ACM Comput. Surv. (CSUR)* **2020**, *53*, 1–42. [CrossRef]
62. Grando, F.; Granville, L.Z.; Lamb, L.C. Machine learning in network centrality measures: Tutorial and outlook. *ACM Comput. Surv. (CSUR)* **2018**, *51*, 1–32. [CrossRef]
63. Sargolzaei, P.; Soleymani, F. Pagerank problem, survey and future research directions. In Proceedings of the International Mathematical Forum, Copenhagen, Denmark, 4–11 July 2004; Citeseer: Roskilde, Denmark, 2010; Volume 5, pp. 937–956.
64. Wang, X.; Bo, D.; Shi, C.; Fan, S.; Ye, Y.; Philip, S.Y. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Trans. Big Data* **2022**, *9*, 415–436. [CrossRef]
65. Zhang, Z.; Wang, X.; Zhu, W. Automated machine learning on graphs: A survey. *arXiv* **2021**, arXiv:2103.00742.
66. Lbath, H.; Bonifati, A.; Harmer, R. Schema inference for property graphs. In Proceedings of the EDBT 2021—24th International Conference on Extending Database Technology, Nicosia, Cyprus, 23–26 March 2021; pp. 499–504.
67. Lutov, A.; Roshankish, S.; Khayati, M.; Cudré-Mauroux, P. Statix—Statistical type inference on linked data. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; IEEE: New York, NY, USA, 2018; pp. 2253–2262.
68. Bouhamoum, R.; Kellou-Menouer, K.; Lopes, S.; Kedad, Z. Scaling up schema discovery for RDF datasets. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW). Paris, France, 16–19 April 2018; IEEE: New York, NY, USA, 2018; pp. 84–89.
69. Pokornỳ, J. Functional querying in graph databases. *Viet. J. Comput. Sci.* **2018**, *5*, 95–105. [CrossRef]
70. Bellaachia, A.; Al-Dhelaan, M. Short text keyphrase extraction with hypergraphs. *Prog. Artif. Intell.* **2015**, *3*, 73–87. [CrossRef]
71. Pokornỳ, J. Graph databases: their power and limitations. In Proceedings of the Computer Information Systems and Industrial Management: 14th IFIP TC 8 International Conference, CISIM 2015, Warsaw, Poland, 24–26 September 2015; Proceedings 14; Springer: Berlin, Germany, 2015; pp. 58–69.
72. Ashmore, R.; Calinescu, R.; Paterson, C. Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. *ACM Comput. Surv.* **2021**, *54*, 111. [CrossRef]
73. Liu, F.; Wu, J.; Xue, S.; Zhou, C.; Yang, J.; Sheng, Q. Detecting the evolving community structure in dynamic social networks. *World Wide Web* **2020**, *23*, 715–733. [CrossRef]
74. Zhang, C.; Song, D.; Huang, C.; Swami, A.; Chawla, N.V. Heterogeneous graph neural network. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 793–803.
75. Agrawal, S.; Jain, S.K. Medical text and image processing: applications, issues and challenges. In *Machine Learning with Health Care Perspective: Machine Learning and Healthcare*; Springer: Cham, Switzerland, 2020; pp. 237–262.
76. Pan, S.J.; Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* **2009**, *22*, 1345–1359. [CrossRef]
77. Shah, F.; Castelltort, A.; Laurent, A. Handling missing values for mining gradual patterns from NoSQL graph databases. *Future Gener. Comput. Syst.* **2020**, *111*, 523–538. [CrossRef]