Software Test Data Management Based on Knowledge Graph

Li Gao*, Junlin Qiu, Guanhua Chen
Faculty of Computer and Software Engineering, Huaiyin Institute of Technology, Huai'an 223003, China
E-mail: gaoli_edu@outlook.com
*Corresponding Author

Keywords: knowledge graph, software testing, data management

Received: June 18, 2024

As software development models and methods mature, large-scale software systems emerge. However, a critical challenge remains: the lack of a comprehensive software test data management model that integrates basic data management with advanced knowledge reasoning. To address this issue, we developed a software test data management model based on knowledge graphs, enabling intelligent management and reasoning of software test data. The model incorporates an entity extraction model based on a feed-forward neural network, a knowledge graph integration method based on graph databases, and a knowledge reasoning submodule based on deep learning. To validate the effectiveness of our model, we evaluated the performance of each component individually. Our deep learning-based entity extraction model achieved an accuracy of 0.92, a recall of 0.88, and an F1 score of 0.90, significantly outperforming traditional methods such as regular expressions and dictionary-based approaches. Utilizing Cypher for graph database querying, our system provides accurate answers with a response time of 0.12 seconds, outperforming SQL and SPARQL-based querying methods. Furthermore, our approach excels in knowledge-based reasoning with an accuracy of 0.89 and site coverage of 0.81, surpassing both ontology-based and graph-based reasoning methods. These results highlight the enhanced construction, querying, and reasoning capabilities of our knowledge graph-based approach for managing software testing data.

Povzetek: Članek opisuje nov model za upravljanje testnih podatkov programske opreme, ki temelji na grafu znanja. Omogoča inteligentno organizacijo, shranjevanje in razumevanje testnih podatkov s pomočjo globokega učenja ter učinkovitejše iskanje in sklepanje v primerjavi s tradicionalnimi metodami.

1 Introduction

Software testing data management refers to the activities of effective organization, storage, maintenance, and utilization of these data, which aims to improve the efficiency and quality of software testing and to reduce the cost and risk of software testing [1]. These activities also support the automation and intelligence of software testing [2]. Software test data management, as an important part of software testing, has been receiving attention from both academia and industry. At present, there have been many methods and tools for software test data management proposed and developed, such as database-based methods, XML-based methods, ontology-based methods, cloud-based methods, etc., and the market share of these methods this year is specifically shown in Figure 1. These methods and tools address some of the challenges of software test data management, such as data normalization, consistency, traceability, reusability, security, etc. to some extent [3].

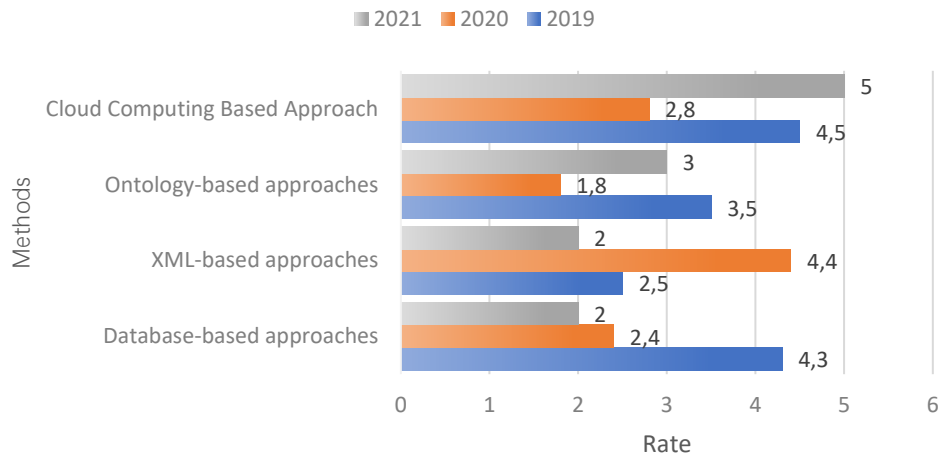

Figure 1: Change in market share of methods and tools for software test data management.

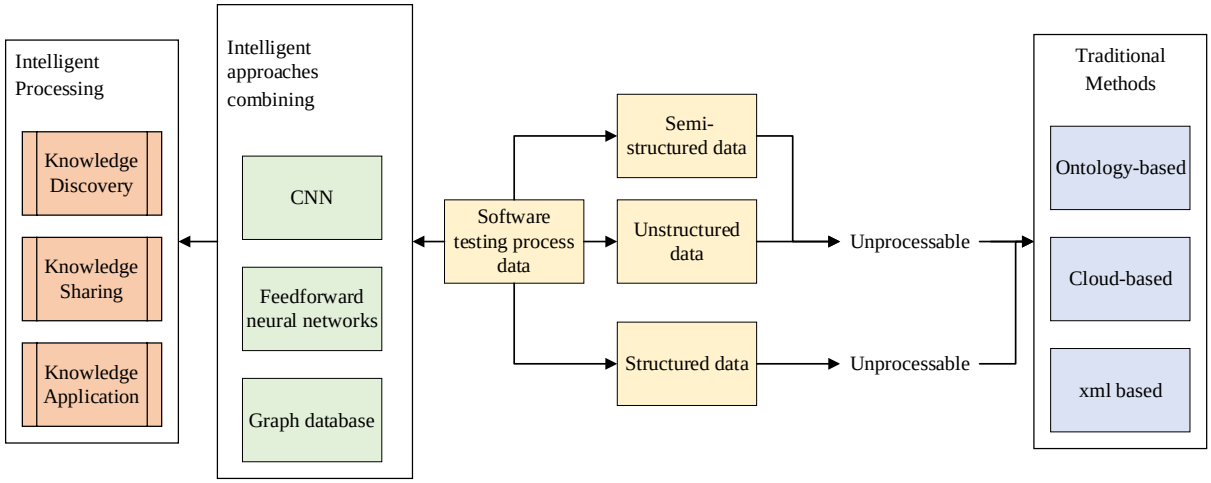

Figure 2: Comparison of approaches to software test data management.

Software testing data contains a lot of unstructured data, and these unstructured data also have important value in software testing, such as for test requirement analysis, test case generation, test result evaluation, etc. On the other hand, traditional tools often can only realize data storage and query, but lack of semantic understanding of data and reasoning ability, which can not meet the intelligent needs of software testing, such as data-based knowledge discovery, knowledge sharing, knowledge application and so on. The comparison between traditional methods of software testing data management and intelligent methods based on knowledge graph is shown in Figure 2 [4, 5].

2 Literature review

Software test data management plays a crucial role in software engineering, which covers a wide range of aspects from acquisition, processing, storage to migration, protection and utilization. This makes the process of data extraction, processing, storage, and migration time-consuming and resource-intensive, as well as increasing technical difficulty and complexity [6, 7]. Secondly, due to the diverse sources of software testing data, such as manually generated, automatically collected, and externally imported, different data sources may have different data standards, formats, contents, and qualities, leading to data inconsistency, which brings difficulties and risks to software testing [8, 9]. In addition, software testing data may contain sensitive information, such as users' personal information, business secrets, etc., which, if leaked or abused, may cause serious losses to users, enterprises and even national security. Therefore, the confidentiality, legality and security of data must be ensured [10]. Finally, there are still some problems in software testing data governance, such as the lack of unified standards and methods, which leads to obstacles and deficiencies in data management and utilization. In order to solve these problems, According to Durst and Zieba [11] suggests the need to optimize and improve the data testing strategy, process, use cases, execution, validation and summarization, and the use of professional testing tools and techniques to assist the data testing process. According to Ebert et al. [12] suggested the use of generative AI techniques to generate large amounts of synthetic data to address issues such as data volume, efficiency, coverage, and privacy, and the use of methods such as data analytics and machine learning to assess and improve data quality. According to Ekanayake et al. [13] emphasized the importance of establishing data governance connotations, elements, models, and frameworks to standardize aspects of data definition, classification, labeling, measurement, monitoring, and evaluation, and to develop data strategies, rules, standards, and processes for effective data management and utilization.In recent years, the construction of knowledge graphs has also made progress, specifically, According to Falát et al. [14] analyzed and sorted out the construction techniques of knowledge graphs and their combination with deep learning; According to Farooq [15] introduced common knowledge graph embedding models and analyzed the prospects of their application in interpretable prediction; These literatures provide valuable references and insights for the theory and application of knowledge graphs.

Knowledge graph has significant advantages in software testing data management. First, it can provide a unified structured representation of structured, semi-structured and unstructured software testing data, and construct a knowledge graph by extracting entities, attributes and relationships to achieve data normalization, consistency and reusability [16, 17]. Secondly, Knowledge Graph can provide semantic annotation and commentary for software testing data, and enhance the semantic information of the data by using knowledge resources such as ontology, lexicon, rules, etc., so as to realize the semanticization, comprehensibility and traceability of the data. In addition, Knowledge Graph can also mine implicit knowledge from data, such as correlation, anomalies, and data evolution through graph algorithms, machine learning, logical reasoning, and other techniques to achieve intelligence, predictability and optimization of data. The structure of knowledge graph is shown in Figure 3 [18, 19].

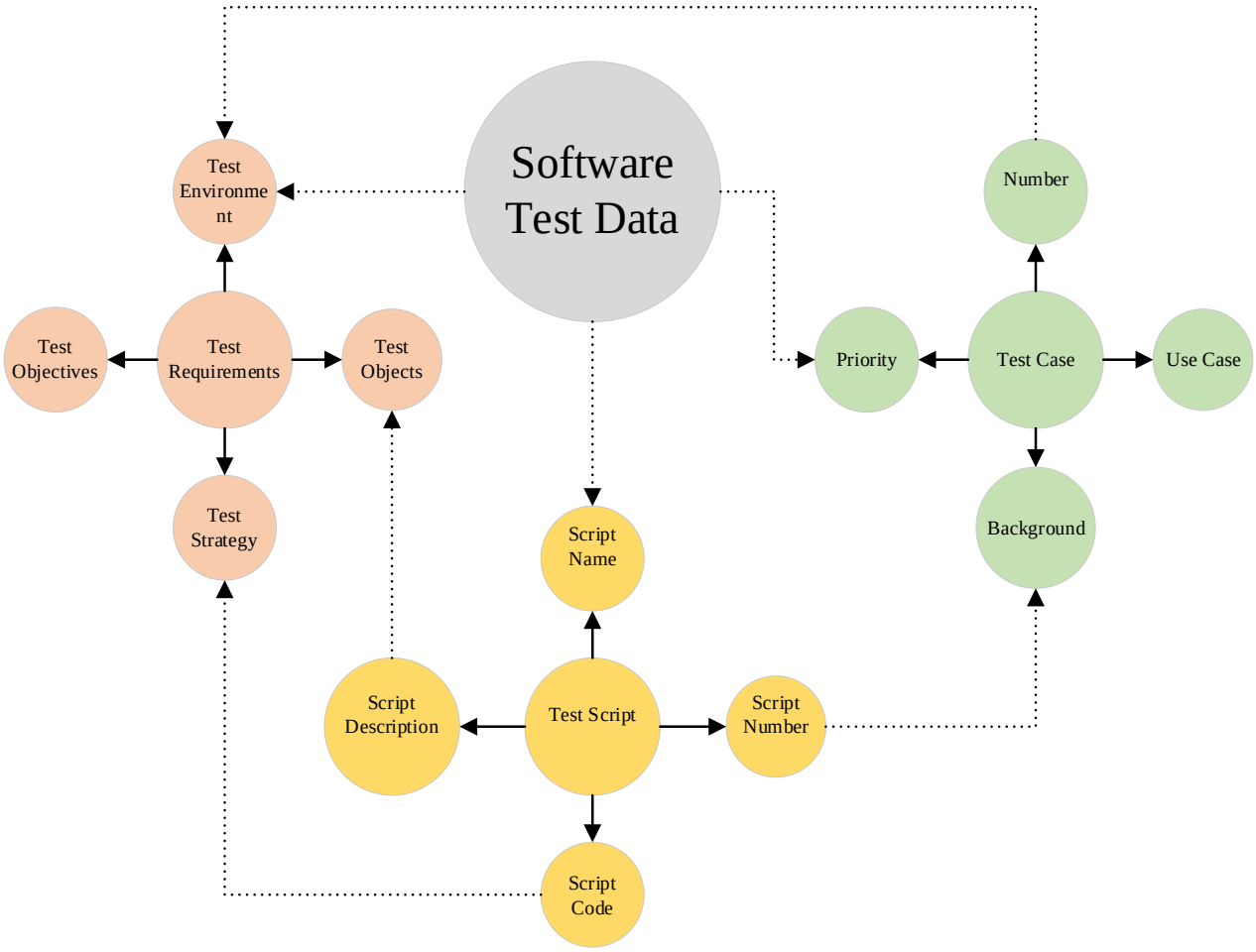

Figure 3: Structure of the knowledge graph.

At present, the research on software test data management based on knowledge graph is still in a blank state. This paper aims to fill this research gap and explore the application value and realization method of knowledge graph in software test data management [20]. The main research content includes designing and realizing a general software test data management knowledge graph model, defining entities, attributes and relationships as well as corresponding ontologies and rules [21]; proposing and realizing the knowledge extraction and knowledge integration process of extracting entities, attributes and relationships from different sources and types of software test data, and integrating them into the knowledge graph [22]; and finally designing and realizing a knowledge integration process that utilizes the knowledge graph's data and knowledge in the Knowledge Graph to design and implement the knowledge query and knowledge reasoning process that provides intelligent support for various aspects of software testing requirement analysis, test case generation, and test result evaluation [23, 24].

To address the challenges in software testing data management, various approaches have been proposed, as summarized in Table 1. Traditional data management relies on manual data handling and basic data storage and retrieval, which are familiar to practitioners and easy to implement but suffer from inconsistent data handling and require significant manual effort [11]. Rule-based data processing provides reliable data extraction for structured data and is easy to define rules but lacks flexibility for unstructured data and limited reasoning capabilities [12]. Dictionary-based data processing is suitable for known entities and offers fast data retrieval but results in incomplete data representation and no semantic enrichment [13].

Recent advances include the use of generative AI techniques for synthetic data generation and augmentation, which addresses data scarcity and enhances data diversity but is limited to data generation and lacks a data management framework [14]. Data analytics and machine learning offer automated insights and scalable processing but do not provide an integrated data management solution or semantic linking [15]. Data governance frameworks standardize data definitions and provide data classification and labeling, ensuring consistent data handling and enhancing data trustworthiness but do not offer a unified data management approach or integration with intelligent systems [16].

Table 1: Comparison of state-of-the-art methods in software testing data management.

Approach	Key contributions	Strengths	Research gap addressed
Traditional data management	Manual data handlingBasic data storage and retrieval	Familiarity among practitionersEase of implementation	Inconsistent data handlingManual effort required
Rule-based data processing	Rule-based data extraction	Reliable for structured dataEasy to define rules	Inability to adapt to new data sourcesLimited reasoning capabilities
Dictionary-based data processing	Dictionary lookup for data categorization	Suitable for known entitiesFast data retrieval	Incomplete data representationNo semantic enrichment
Generative AI techniques	Synthetic data generationData augmentation	Addresses data scarcityEnhances data diversity	Limited to data generationNo data management framework
Data analytics and machine learning	Data quality assessmentAutomated anomaly detection	Automated insights Scalable processing	No integrated data management solutionNo semantic linking
Data governance frameworks	Standardization of data definitionsData classification and labeling	Consistent data handlingEnhanced data trustworthiness	No unified data management approachNo integration with intelligent systems
Knowledge graph-based approach (proposed)	Unified structured representation of dataSemantic annotation and enrichmentIntelligent reasoning	Comprehensive data managementSemantic consistencyIntelligent support	Unified data management framework Integration of data and knowledgeIntelligent reasoning

3 Knowledge graph-based software test data management modeling

This chapter details our proposed and innovative knowledge graph-based software testing data management model [25]. The core idea of the model is to utilize the powerful expression and reasoning ability of knowledge graph to effectively organize, manage and apply all kinds of data in the software testing process. Its principle is mainly to transform all kinds of complex test data into forms that are easy to understand and process by constructing a knowledge graph containing software testing related knowledge, so as to realize intelligent management of test data [26, 27].

3.1 Modeling ideas

The model idea of this paper is to consider the process of software test data management as a process of constructing and applying a knowledge graph, i.e., extracting entities, attributes and relationships from software test data, constructing a knowledge graph for software test data management, and then utilizing the data and knowledge in the knowledge graph to provide intelligent support for software testing. The modeling idea of this paper is based on the following facts:

Firstly, software testing data contains rich knowledge, such as software testing requirements, use cases, results, etc., which can be represented in the form of entities, attributes and relationships, constituting a knowledge graph for software testing data management. Secondly, the knowledge graph of software testing data management can be structured, semanticized and intelligently processed to improve the quality and value of software testing data and provide effective support for all aspects of software testing [28]. Finally, the knowledge graph for software testing data management can be constructed from multi-source heterogeneous software testing data by knowledge extraction and knowledge integration methods, and can be intelligently applied by knowledge query and knowledge reasoning methods [29, 30].

3.2 Modeling framework

In this paper, we propose a framework for a software test data management model based on knowledge graph, as shown in Figure 4. The framework includes three main modules: knowledge graph construction module, knowledge graph storage module and knowledge graph application module [31].

The knowledge graph building module is the module responsible for extracting and integrating entities, attributes and relationships from software test data to build a knowledge graph for software test data management.

The module includes two submodules: entity extraction submodule and knowledge graph integration submodule. The entity extraction submodule is a submodule that recognizes entities and their related attributes from different types and sources of software testing data using deep learning-based entity extraction methods [32]. The knowledge graph integration submodule is a submodule that utilizes a graph database-based knowledge graph integration approach to integrate entities, attributes and relationships extracted from software testing data into a unified knowledge graph for software testing data management [33].

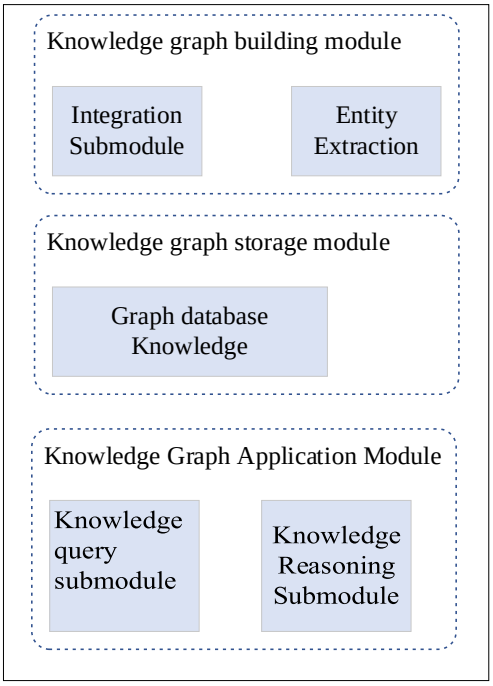

Figure 4: Framework of the software test data management model.

The Knowledge Graph Storage Module is the module responsible for storing and managing the knowledge graph of software test data management. The module uses a graph database as a storage for the knowledge graph, and utilizes the characteristics of a graph database, such as nodes, edges, labels, and attributes, to represent the entities, attributes, and relationships in the knowledge graph for software test data management, as well as the structure and semantics between them.

The Knowledge Graph Application Module is the module responsible for providing intelligent support for all aspects of software testing by utilizing the knowledge graph of software testing data management. The module consists of two submodules: the knowledge query submodule and the knowledge reasoning submodule. The knowledge query submodule is a submodule that utilizes a query language for graphical databases, such as Cypher, to query the data and knowledge in the knowledge graph of software testing data management, and to achieve data retrieval and analysis. The Knowledge Reasoning submodule is a submodule that utilizes knowledge reasoning techniques, such as rule-based knowledge reasoning, graph-based knowledge reasoning, and learning-based knowledge reasoning, to derive implicit knowledge from the knowledge graph of software testing data management, and to realize knowledge discovery and application [34-36].

3.3 Modeling principles

The model principle of this paper is based on the technique of knowledge graph, including knowledge extraction, knowledge integration, knowledge query and knowledge reasoning, to realize the construction and application of knowledge graph for software testing data management. The model schematic is specifically shown in Figure 5.

Figure 5 shows a process of concept recognition and diagram construction. First of all, the input is "test data", after "encoder" processing to get "sentence vector". The sentence vector then interacts with the graph vector via a multilayer perceptron (MLP) to generate a feasibility score. At the same time, "sentence vectors" are also used for "concept recognition" and further converted into nodes in "graph construction". Finally, in the process of graph construction, a "pattern graph" is created using pathfinding methods. Convolutional neural networks (CNNs) may play a role in some aspects of this process.

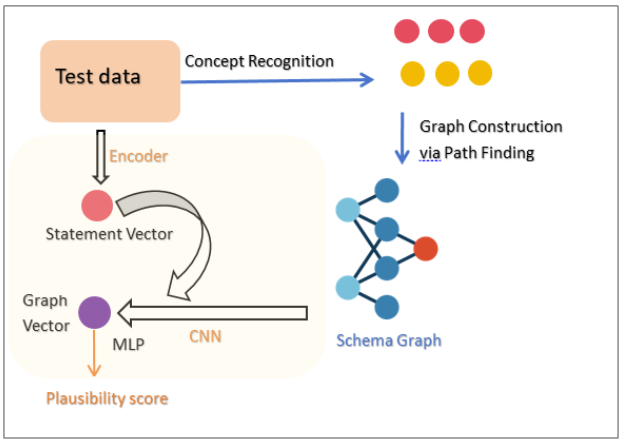

Figure 5: Model schematic.

In our pursuit to automate entity extraction and knowledge integration for the creation of a structured knowledge graph, we have developed a sophisticated deep learning-based approach. This methodology leverages the strengths of both Convolutional Neural Networks (CNNs) and Recurrent Neural Networks, specifically Long Short-Term Memory (LSTM) networks, to process and analyze textual data with the aim of extracting entities and their relationships from text. The architecture of our entity extraction model is meticulously designed, starting with text preprocessing to clean and tokenize the data, followed by the conversion of words into dense vector representations through word embeddings. CNNs are then applied to capture local features within sentences, with Bi-LSTM layers subsequently employed to understand the long-range dependencies within the text. To ensure consistent entity tagging, we incorporate a Conditional Random Field (CRF) layer. The training process involves preparing labeled data, initializing the model with pre-

trained embeddings, and iteratively refining the model through mini-batch training, loss calculation, and backpropagation. Model evaluation is conducted using precision, recall, and F1-score metrics, with hyperparameter tuning to optimize performance. Post-entity extraction, we proceed with knowledge integration, which includes entity linking to the knowledge graph, relationship extraction, and graph construction. This integrated approach not only yields high accuracy in entity recognition but also facilitates the construction of a comprehensive and informative knowledge graph tailored for software testing data management.

3.3.1 Entity extraction submodule

In this paper, a deep learning based entity extraction approach is used to automatically extract entities and their related attributes from software test data using feed forward neural network models. It is based on the principle of setting D as software test data, E as entity, A as attribute, R as relationship, M as entity extraction model, T as data type, S as data source, O as output, F as feature, C as context, L as semantics, P as attention mechanism, B as bidirectional recurrent neural network, X as convolutional neural network, and Z as pre-trained language model, and then it can be expressed by Equation 1 [37].

$$\begin{aligned} O &= M(D) \\ M(D) &= E(D) \cup A(D) \cup R(D) \\ M(D) &= \begin{cases} M_B(D) & \text{if } \mathrm{T}(D) = Structuring. \\ M_X(D) & \text{if } \mathrm{T}(D) = Semi-structured. \\ M_Z(D) & \text{if } \mathrm{T}(D) = Unstructured \end{cases} \\ M_B(D) &= F(D) \cdot P(C(D)) \\ M_X(D) &= F(D) \cdot X \\ M_Z(D) &= F(D) \cdot Z(L(D)) \end{aligned} \tag{1}$$

3.3.2 Knowledge integration submodule

In this paper, we adopt a knowledge graph integration method based on graphical databases, which utilizes the characteristics of graphical databases to store and manage the knowledge graph for software test data management. For storing entities, this paper uses a graph database node to represent entities, each node contains a unique identifier (ID), one or more labels, and one or more Property-Value Pairs. For storing relationships, this paper uses edges from a graph database to represent relationships, where each edge contains a unique identifier (ID), a Type, and one or more Property-Value Pairs. In this paper, we use Cypher as the query language, which is a pattern-matching based query language for graph databases that can easily represent query patterns for graph structures, as well as operations such as filtering and aggregation of query results [38].

3.3.3 Knowledge reasoning submodule

CNN-based knowledge inference submodule is another important component in the application of knowledge graph-based software test data management model, and its main function is to reason out the unknown data based on the existing data in the knowledge graph, so as to complement and extend the knowledge graph. The basic principle of CNN-based knowledge inference submodule is as follows:

For example, for the entity “Selenium” and the relationship “support”, it can be shown in Equation 2.

$$\begin{aligned} Selenium &= [0.2,\ -0.5,\ 0.7,\ 0.1] \\ support &= [-0.3,\ 0.4,\ -0.6,\ 0.2] \end{aligned} \tag{2}$$

For example, for the triad (Selenium, support, Automation Testing), the reasonableness can be calculated using a score function as shown in Equation 3:

$$\begin{aligned} &score(Selenium,\ support,\ AutomationTesting) \\ &= f(Selenium\ +\ support\ -\ AutomationTesting) \end{aligned} \tag{3}$$

Where f is a nonlinear activation function such as sigmoid or tanh for mapping the score to a fixed interval such as [0, 1] or [-1, 1]. The model parameters can be optimized with a loss function as shown in Equation 4 for the known triad (Selenium, Support, Automation Testing) and the unknown triad (Selenium, Support, Unit Testing).

$$loss = max\begin{pmatrix} 0,\ gamma\ +\ score\begin{pmatrix} Selenium,\ support, \\ AutomationTesting \end{pmatrix} \\ -\ score(Selenium,\ support,\ UnitTesting) \end{pmatrix} \tag{4}$$

4 Evaluation and validation of models

In this paper, two publicly available datasets are used to evaluate the performance and effectiveness of the knowledge graph-based software testing data management model, namely (1) Software Testing Data Set. This dataset contains data from software testing projects from 2006 to 2014, including test requirements, test cases, test results, test defects, etc., with nine tables and about 100,000 records. This dataset can be used to construct a knowledge graph for software testing data management, as well as experiments for knowledge query and reasoning. (2) Software Engineering Data Set: This dataset contains the data of software engineering projects from 2010 to 2018, including software requirements, software design, software code, software defects, etc. This dataset can be used for experiments of knowledge fusion and extension with the knowledge graph of software test data management.

The Software Testing Data Set, gathered from real-world projects between 2006 and 2014, encompasses approximately 100,000 records across nine tables, detailing test requirements, cases, results, and defects, along with additional contextual data. It's utilized to build

a knowledge graph for testing data management and supports research in knowledge query and reasoning. Similarly, the Software Engineering Data Set, spanning 2010 to 2018, contains around 200,000 records in 12 tables, covering software requirements, design, code, and defects, plus extra metadata. It facilitates knowledge fusion and extends the testing data management knowledge graph, aiding in integrating testing with broader software engineering processes. Both datasets are publicly accessible and have been pivotal in academic research, ensuring the reproducibility of experimental results. Researchers can access these datasets through the provided links, adhering to the respective data usage guidelines.

In this paper, the following evaluation metrics are used to measure the performance and effectiveness of the knowledge graph-based software test data management model, which are accuracy of knowledge graph construction, recall of knowledge graph construction, F1 value of knowledge graph construction, accuracy of knowledge query, response time of knowledge query Accuracy of knowledge reasoning, coverage of knowledge reasoning [39].

In this paper, a deep learning-based entity extraction method and a graph database-based knowledge graph integration method are used to construct a knowledge graph for software test data management from a software test dataset. The experimental results are shown in Table 2.

Table 2: Experimental results of knowledge graph construction.

Methodologies	Accuracy	Recall rate	F1 value
Deep learning based approach	0.92	0.88	0.90
Regular expression based approach	0.78	0.71	0.74
Dictionary-based approach	0.68	0.63	0.65

As can be seen from Table 2, the deep learning-based approach outperforms the other two rule-based approaches in terms of accuracy, recall, and F1 value of knowledge graph construction, indicating that the deep learning-based approach can more effectively extract entities, attributes, and relationships from software test data and construct a more complete and accurate knowledge graph [40].

In order to validate the capability of the system for knowledge querying, this paper uses a graph database based query language, such as Cypher, to retrieve relevant answers from the knowledge graph of software test data management. In this paper, 20 natural language questions of different types and difficulty, involving entity queries, relational queries, path queries, and aggregation queries, are designed as a test set. In this paper, the accuracy and response time of the knowledge queries are calculated using the correct answers given manually as a reference. The experimental results are shown in Table 2.

As can be seen from Table 3, the graph database-based query language outperforms the other two relational database-based query languages in terms of knowledge query accuracy and response time, which indicates that the graph database-based query language can retrieve the relevant answers from the knowledge graph of software test data management more efficiently, and improves the efficiency and quality of the retrieval [41, 42].

In this paper, we have used a part of the data from the software test dataset as a training set and another part of the data as a test set for training knowledge-based reasoning models and evaluating the effectiveness of knowledge-based reasoning, respectively. In this paper, the accuracy and coverage of knowledge reasoning is calculated using the correct knowledge given manually as a reference. This paper is also compared with ontology-based knowledge reasoning and graph-based knowledge reasoning, and the results are shown in Table 3.

Table 3: Experimental results of knowledge query.

Methodologies	Accuracy	Response time (seconds)
A query language based on graph databases	0.95	0.12
Relational database based SQL	0.85	0.23
SPARQL based on relational databases	0.80	0.28

Table 4: Experimental results of knowledge-based reasoning.

Methodologies	Accuracy	Site coverage
Deep learning based approach	0.89	0.81
Ontology-based approach	0.75	0.68
A graph-based approach	0.69	0.62

As can be seen from Table 4, the deep learning-based approach outperforms the other two rule-based approaches in terms of accuracy and coverage of knowledge inference, indicating that the deep learning-based approach can be more effective in inference of unknown knowledge from the knowledge graph of software test data management, and in complementing and expanding the knowledge graph [43].

5 Discussion

5.1 Entity extraction and knowledge graph construction

The proposed deep learning-based entity extraction method outperforms the rule-based approaches, as shown in Table 5. The deep learning approach achieves higher accuracy, recall, and F1 value in constructing the

knowledge graph, demonstrating its effectiveness in extracting entities, attributes, and relationships from software testing data.

Table 5: Comparison of entity extraction and knowledge graph construction.

Method	Accuracy	Recall rate	F1 value
Deep Learning-Based Approach	0.92	0.88	0.90
Regular Expression-Based	0.78	0.71	0.74
Dictionary-Based	0.68	0.63	0.65

The superior performance of the deep learning-based approach can be attributed to its ability to learn complex patterns and relationships within the data, making it more adaptable to variations in the input data.

5.2 Knowledge query and reasoning

The graph database-based query language demonstrates superior performance over traditional relational database query languages, as illustrated in Table 5. The graph database approach achieves higher accuracy and faster response times, which are critical for efficient and high-quality retrieval of information from the knowledge graph.

Table 6: Comparison of knowledge query and reasoning.

Method	Accuracy	Response time (Seconds)
Graph database-based	0.95	0.12
Relational database (SQL)	0.85	0.23
SPARQL (Relational DB)	0.80	0.28

The graph database-based query language leverages the inherent structure of the knowledge graph, enabling faster and more precise query execution. Additionally, the deep learning-based approach to knowledge reasoning outperforms both the ontology-based and graph-based approaches, as shown in Table 6. The deep learning method achieves higher accuracy and site coverage, indicating its effectiveness in inferring unknown knowledge and complementing the existing knowledge graph.

Table 7: Comparison of knowledge reasoning.

Method	Accuracy	Site Coverage
Deep learning-based	0.89	0.81
Ontology-based	0.75	0.68
Graph-based	0.69	0.62

As shown in Table 7, the deep learning-based approach benefits from its ability to learn from patterns and relationships in the data, allowing it to make more accurate inferences and expand the knowledge base.

5.3 Novelty and benefits of the proposed method

The proposed knowledge graph-based software testing data management model introduces a novel and beneficial approach by offering comprehensive data management through a unified structured representation, semantic enrichment for enhanced data understanding, intelligent support for inferring new knowledge and aiding decision-making, and efficient querying via a graph database query language, collectively addressing the prevalent challenges in software testing data management and showcasing its unique value.

6 Conclusion

This paper proposes a software testing data management model based on knowledge graph, which can realize intelligent management and reasoning of software testing data. The model consists of three submodules, which are feed-forward neural network-based entity extraction module, graph database-based knowledge graph integration module, and deep learning-based knowledge inference module. The model provides a new idea and method for software testing data management, which helps to improve the efficiency and quality of software testing.

Funding

The research is supported by Jiangsu Province Industry-University-Research Cooperation Project, Research and Development of a Smart Constitute Site Safety Management System (No: BY20221107).
Conflict of interest: The authors state no conflict of interest.

References

[1] Ahmad, T., Iqbal, J., Ashraf, A., Truscan, D., & Porres, I. Model-based testing using UML activity diagrams: A systematic mapping study. Computer Science Review, 33, 98-112, 2019. https://doi.org/10.1016/j.cosrev.2019.07.001.

[2] Alyahya, S. Collaborative crowdsourced software testing. Electronics, 11(20), 3340, 2022. https://doi.org/10.3390/electronics11203340.

[3] Anthony, B. Information flow analysis of a knowledge mapping-based system for university alumni collaboration: A practical approach. Journal of the Knowledge Economy, 12(2), 756-787, 2021. https://doi.org/10.1007/s13132-020-00643-3.

[4] Ben Zayed, H. A., & Maashi, M. S. Optimizing the software testing problem using search-based software engineering techniques. Intelligent Automation and Soft Computing, 29(1), 307-318, 2021. https://doi.org/10.32604/iasc.2021.017239.

[5] Benhar, H., Idri, A., & Fernández-Alemán, J. L. A systematic mapping study of data preparation in heart disease knowledge discovery. Journal of Medical Systems, 43, 1-17, 2019. https://doi.org/10.1007/s10916-018-1134-z.

[6] Boopathi, M., Sujatha, R., Kumar, C. S., Narasimman, S., & Rajan, A. Markov approach for quantifying the software code coverage using genetic algorithm in software testing. International Journal of Bio-Inspired Computation, 14(1), 27-45, 2019. https://doi.org/10.1504/ijbic.2019.101152.

[7] Calvanese, D., Gal, A., Lanti, D., Montali, M., Mosca, A., & Shraga, R. Conceptually grounded mapping patterns for virtual knowledge graphs. Data & Knowledge Engineering, 145, 102157, 2023. https://doi.org/10.1016/j.datak.2023.102157.

[8] Chen, T., Zhang, S. J., Wang, Y., Chen, Z. B., & Jing, W. F. Construction methods of knowledge mapping for full service power data semantic search system. Journal of Signal Processing Systems for Signal Image and Video Technology, 93, 275-284, 2021. https://doi.org/10.1007/s11265-020-01591-6.

[9] Cordeiro, M., Puig, F., & Ruiz-Fernández, L. Realizing dynamic capabilities and organizational knowledge in effective innovations: the capabilities typological map. Journal of Knowledge Management, 27(10), 2581-2603, 2022. https://doi.org/10.1108/jkm-02-2022-0080.

[10] Drave, I., Hillemacher, S., Greifenberg, T., Kriebel, S., Kusmenko, E., Markthaler, M., Orth, P., Salman, K. S., Richenhagen, J., & Rumpe, B. SMArDT modeling for automotive software testing. Software-Practice & Experience, 49(2), 301-328, 2019. https://doi.org/10.1002/spe.2650.

[11] Durst, S., & Zieba, M. Mapping knowledge risks: Towards a better understanding of knowledge management. Knowledge Management Research & Practice, 17(1), 1-13, 2019. https://doi.org/10.1080/14778238.2018.1538603.

[12] Ebert, C., Bajaj, D., & Weyrich, M. Testing software systems. IEEE Software, 39, 8-17, 2022. DOI: 10.1109/ms.2022.3166755.

[13] Eisty, N. U., & Carver, J. C. Testing research software: A survey. Empirical Software Engineering, 27, 28, 2022. https://doi.org/10.1007/s10664-022-10184-9.

[14] Ekanayake, E., Shen, G., & Kumaraswamy, M. M. Mapping the knowledge domains of value management: A bibliometric approach. Engineering construction and Architectural Management, 26(3), 499-514, 2019. https://doi.org/10.1108/ecam-06-2018-0252.

[15] Falát, L., Michalová, T., Madzík, P., & Marsíková, K. Discovering trends and journeys in knowledge-based human resource management: Big data smart literature review based on machine learning approach. IEEE Access, 11, 95567-95583, 2023. https://doi.org/10.1109/access.2023.3296140.

[16] Farooq, R. Knowledge management and performance: A bibliometric analysis based on Scopus and WOS data (1988-2021). Journal of Knowledge Management, 27(7), 1948-1991, 2023. https://doi.org/10.1108/jkm-06-2022-0443.

[17] Foidl, H., & Felderer, M. Integrating software quality models into risk-based testing. Software Quality Journal, 26, 809-847, 2018. https://doi.org/10.1007/s11219016-9345-3.

[18] Garousi, V., Bauer, S., & Felderer, M. NLP-assisted software testing: A systematic mapping of the literature. Information and Software Technology, 126, 106321, 2020. https://doi.org/10.1016/j.infsof.2020.106321.

[19] Garousi, V., Felderer, M., Karapiçak, Ç., & Yilmaz, U. Testing embedded software: A survey of the literature. Information and Software Technology, 104, 1445, 2018. https://doi.org/10.1016/j.infsof.2018.06.016.

[20] Garousi, V., Felderer, M., & Kiliçaslan, F. N. A survey on software testability. Information and Software Technology, 108, 35-64, 2019. https://doi.org/10.1016/j. infsof.2018.12.003.

[21] Garousi, V., Felderer, M., Kuhrmann, M., Herkiloglu, K., & Eldh, S. Exploring the industry's challenges in software testing: An empirical study. Journal of Software-Evolution and Process, 32(8), e2251, 2020. https://doi.org/10.1002/smr.2251.

[22] Garousi, V., & Küçük, B. Smells in software test code: A survey of knowledge in industry and academia. Journal of Systems and Software, 138, 52-81, 2018. https://doi.org/10.1016/j.jss.2017.12.013.

[23] Garousi, V., Rainer, A., Lauvås, P., & Arcuri, A. Software-testing education: A systematic literature mapping. Journal of Systems and Software, 165, 110570, 2020. https://doi.org/10.1016/j.jss.2020.110570.

[24] Ho, V. W., Harris, P. G., Kumar, R. K., & Velan, G. M. Knowledge maps: A tool for online assessment with automated feedback. Medical Education Online, 23(1), 1457394, 2018. https://doi.org/10.1080/10872981.2018.1457394.

[25] Huang, T., & Fang, C. C. Optimization of software test scheduling under development of modular software systems. Symmetry-Basel, 15(1), 195, 2023. https://doi.org/10.3390/ sym15010195.

[26] Huang, Y., Glänzel, W., & Zhang, L. Tracing the development of mapping knowledge domains. Scientometrics, 126, 6201-6224, 2021. https://doi.org/10.1007/s11192-02003821-x.

[27] Idri, A., Benhar, H., Fernández-Alemán, J. L., & Kadi, I. A systematic map of medical data preprocessing in knowledge discovery. Computer Methods and Programs in Biomedicine, 162, 69-85, 2018. https://doi.org/10.1016/j.cmpb.2018.05.007.

[28] Jung, P., Kang, S., & Lee, J. Automated code-based test selection for software product line regression testing. Journal of Systems and Software, 158,

110419, 2019. https://doi.org/10.1016/j.jss.2019.110419.
[29] Jung, P., Kang, S., & Lee, J. Efficient regression testing of software product lines by reducing redundant test executions. Applied Sciences-Basel, 10(23), 8686, 2020. https://doi.org/10.3390/app10238686.
[30] Kaur, V. Knowledge-based dynamic capabilities: A scientometric analysis of marriage between knowledge management and dynamic capabilities. Journal of Knowledge Management, 27(4), 919-952, 2023. https://doi.org/10.1108/jkm-02-2022-0112.
[31] Khan, M. U., Sherin, S., Lqbal, M. Z., & Zahid, R. Landscaping systematic mapping studies in software engineering: A tertiary study. Journal of Systems and Software, 149, 396-436, 2019. https://doi.org/10.1016/j.jss.2018.12.018.
[32] Laaber, C., Gall, H. C., & Leitner, P. Applying test case prioritization to software microbenchmarks. Empirical Software Engineering, 26(6), 133, 2021. https://doi.org/10.1007/s10664-021-10037-x.
[33] Lee, J., Kang, S., & Keum, C. Architecture-Based software testing. International Journal of Software Engineering and Knowledge Engineering, 28(1), 57-77, 2018. https://doi.org/10.1142/s0218194018500031.
[34] Lee, J. H. Mapping local knowledge through spatial text mining. Landscape and Ecological Engineering, 19(2), 243-255, 2023. https://doi.org/10.1007/s11355-023-005411.
[35] De, R., & Nanda, I. Network/security threats and countermeasures for cloud computing. Acta Electronica Malaysia, 7(1), 1-3, 2022. https://doi.org/10.26480/aem.01.2022.01.03
[36] Rachman, A., Kurniawan, M., Anam, C., Putra, R. E., Rozi, N.F., Sulistyowati, & Pakarbudi, A. Fast development kangean island tourism website using maf-inc model. Acta Informatica Malaysia, 7(2), 83-91, 2023. https://doi.org/10.26480/aim.02.2023.83.91
[37] Liargkovas, G., Papadopoulou, A., Kotti, Z., & Spinellis, D. Software engineering education knowledge versus industrial needs. IEEE Transactions on Education, 65(3), 419-427, 2022. https://doi.org/10.1109/te.2021.3123889.
[38] Marculescu, B., Feldt, R., Torkar, R., & Poulding, S. Transferring interactive search-based software testing to industry. Journal of Systems and Software, 142, 156-170, 2018. https://doi.org/10.1016/j.jss.2018.04.061.
[39] Menaouer, B., & Nada, M. The relationship between knowledge mapping and the open innovation process: The case of education system. AI Edam-Artificial Intelligence for Engineering Design Analysis and Manufacturing, 34(1), 17-29, 2020. https://doi.org/10.1017/s0890060419000325.
[40] Peischl, B., Tazl, O. A., & Wotawa, F. Testing anticipatory systems: A systematic mapping study on the state of the art. Journal of Systems and Software, 192, 111387, 2022. https://doi.org/10.1016/j.jss.2022.111387.
[41] Pellegrini, M. M., Ciampi, F., Marzi, G., & Orlando, B. The relationship between knowledge management and leadership: Mapping the field and providing future research avenues. Journal of Knowledge Management, 24(6), 1445-1492, 2020. https://doi.org/10.1108/jkm-01-2020-0034.
[42] Odeh, A. Exploring AI innovations in automated software source code generation: Progress, hurdles, and future paths. Informatica, An International Journal of Computing and Informatics, 48(8), 125-136, 2024. https://doi.org/10.31449/inf.v48i8.5291.
[43] Sofian, H., Yunus, N. A. M., & Ahmad, R. Systematic mapping: Artificial intelligence techniques in software engineering. IEEE Access, 10, 51021-51040, 2022. https://doi.org/10.1109/access.2022.3174115.