Mesopotamian journal of Big Data
Vol. **(2025)**, 2025, **pp**. 394–414
DOI: https://doi.org/10.58496/MJBD/2025/024 , ISSN: 2958-6453
https://mesopotamian.press/journals/index.php/BigData

Research Article

A Novel AI-based Dependency-aware Algorithm for Prioritizing Software Requirements in Large-scale Projects

Nahla Mohamed[1, *], Waleed Helmy[1], Sherif Mazen [1]

[1]*Department of Information Systems, Faculty of Computing and Artificial Intelligence, Cairo University, Giza, Egypt*

ARTICLEINFO

Article History

Received 19 Jul 2025
Revised 31 Aug 2025
Accepted 13 Sep 2025
Published 29 Oct 2025

Keywords

Requirement engineering
Requirement prioritization
Requirement dependency analysis
Analytical hierarchy process
Large language models

ABSTRACT

Requirement prioritization (RP) is one of the main activities in software analysis; an incorrect RP process can lead to many software failures. In any software project, the requirements are interdependent. Most current RP techniques almost overlook the requirements dependency (RD) handling while prioritizing. Neglecting dependencies among requirements during the RP task can lead to deadlock and incorrect prioritization results, resulting in high rework and project delays. This motivates us to introduce a novel, scalable, dependency-aware RP algorithm, namely, then Dependency-Aware Enhanced Analytical Hierarchy Process (DA-EAHP), which integrates an RD handling mechanism using fine-tuned large language models (LLMs) into our previously developed RP technique, namely, the Enhanced Analytical Hierarchy Process (E-AHP), to increase the realism and accuracy of the software RP process. The proposed algorithm is assessed against two zero-shot-based LLM models and a fuzzy graph-based model. All were evaluated on various-sized subsets from the PURE dataset, ranging from 25 500 requirements, to compare their dependency analysis accuracy and computational performance. The results show that our proposed algorithm achieves a 7–18% accuracy improvement over the baselines, with an approximate reduction of 52–82% and 71–79% in time and memory, respectively. Moreover, a comparison with another variant of the proposed algorithm without the RD handling process validated its positive impact on the RP process. These results show that DA-EAHP provides a more accurate and efficient RP technique, making it suitable for large-scale software projects with complicated dependencies. Research limitations include the dependence on scalability beyond 500 requirements and expert-annotated ground truth, which are open for future work.

1. INTRODUCTION

Requirement prioritization (RP) plays a critical role in the success of any software project, especially agile projects [1-3]. Specifying essential requirements within a specified time, budget, and user-satisfaction constraints requires a highly structured and scalable RP process [2]. An inaccurate RP process can lead to high rework, high costs, and client dissatisfaction, resulting in project failure [4,5]. One of the main challenges in the RP process is handling requirement dependencies (RDs) during prioritization [5-11]. As in any software project, the requirements are interrelated, and implementing one often requires implementing another first. Ignoring these RDs can lead to bottleneck development, inaccurate prioritization results, deadlocks, and inefficiency in resource allocation. Moreover, some dependencies cause cascading effects, where the failure or delay of one can affect multiple other downstream requirements. Therefore, a high-quality RP process must consider RD while prioritizing.

Existing approaches and techniques for RP and RD detection have achieved partial success. For example, large language models (LLMs) excel in capturing semantic context and meaning, providing high accuracy in complex scenarios. Fuzzy-based models offer flexibility for uncertainty handling and interpretability. Natural language processing (NLP) and machine learning (ML) offer efficiency and scalability, especially with moderate-sized and structured datasets. These successes highlight the strengths of various techniques in solving RP challenges. However, different limitations exist. For example, LLMs require high computational resources and domain-specific fine-tuning. Fuzzy methods depend on predefined membership functions, limiting their adaptability to various domains. Classical NLP and ML approaches cannot capture nuanced contextual RDs. These limitations explain why current approaches have not become optimal solutions that balance scalability, interpretability, and accuracy, highlighting the need for more improvements.

Corresponding author. Email: n.abdellatif@fci-cu.edu.eg

Our survey research [12] shows that limitations exist in the current RD analysis approaches and RP techniques, including the following:

- Most RP techniques face serious limitations regarding RD handling, as almost none handle RD while prioritizing.
- Most current RD analysis approaches have not been tested on real or large datasets, raising concerns about their robustness, scalability, and applicability in real software environments.
- Although artificial intelligence-based RD analysis approaches, specifically LLMs, prove performance superiority in implicit RD detection, they lack transparency, as they function as black boxes. They also require high computational resources, limiting their standalone applicability.
- In contrast, traditional rule-based approaches, especially graph-based techniques, offer better explainability but often face difficulty in detecting hidden and semantic RDs.

These limitations emphasize the need for a new dependency-aware, scalable, transparent RP approach to handle large requirements, which motivates us to enhance our previously proposed RP technique, namely, the Enhanced Analytical hierarchy process (E-AHP) [13], by introducing a dependency-aware E-AHP (DA-EAHP) that extends it with a scalable, accurate, transparent dependency-aware mechanism using strong structure rules, LLMs, and NLP to be a comprehensive RP technique that enhances the decision-making process in requirements engineering.

The main contributions of this research paper can be summarized as follows:

- A dependency-aware RP technique, namely, DA-EAHP, extends our previously proposed E-AHP algorithm to consider RD while prioritizing by integrating semantic similarity analysis using LLMs to extract implicit and complex RD, enhancing RP's contextual awareness, and adjusting the RP process result by considering the dependency relationships.
- An automatic domain-specific requirements scoring mechanism that replaces user input in E-AHP with predefined domain-specific keywords to determine a more accurate score for each requirement, and enhances scalability.
- Replacing heuristic grouping in E-AHP with a rule-based grouping mechanism that groups the requirements according to structural cues enables an effective and scalable RP process.
- A comprehensive evaluation of our proposed algorithm's performance, scalability, and accuracy in handling large requirements and considering RD against zero-shot LLM and recent fuzzy graph-based models using a PURE dataset, which was selected owing to its popularity in the requirements engineering (RE) field and its domain relevance.
- The full proposed algorithm and a variant without the RD analysis phase are compared to highlight the added value and impact of integrating the RD handling task into the RP process.

Therefore, the paper's objective is to propose a comprehensive dependency-aware RP algorithm that enhances the scalability and accuracy of RP when handling large requirements with a complex dataset. The originality of this paper lies in incorporating a dependency analysis mechanism into an enhanced version of the AHP framework, enabling it to handle RD effectively. Unlike others, which focus only on the RD or RP task, our proposed algorithm offers a novel, unique solution by integrating both with high accuracy, making our algorithm an effective and comprehensive RP approach. By considering the RD in the prioritization framework, the algorithm can prevent costly rework and minimize sequencing errors, leading to more consistent and correct decisions and supporting the delivery of high-quality software products.

The rest of the paper is structured as follows: Section 2 presents an overview of related works on RD management approaches and techniques. Section 3 presents the research background and main concepts. Section 4 introduces our proposed algorithm, DA-EAHP. Section 5 presents the experimental setup, Section 6 presents the results and discussion, Section 7 discusses the study's limitations and threats to validity, Section 8 concludes the paper, Section 9 outlines future work, and Section 10 presents the data and code availability.

2. RELATED WORKS

This section reviews existing techniques proposed to detect software RDs, highlighting their advantages and limitations. Researchers [14] use an NLP-based technique to determine and extract dependencies among requirements. It first applies NLP to the collected software requirements in the SRS. After that, it analyses the main terms as a semantic structure. This approach focuses mainly on the evolution of the requirements dependencies. This tool dynamically reassesses dependencies, ensuring the accuracy of the discovered dependency. Instead, it incrementally evaluates the updates, which improves adaptability. Although this tool succeeded in supporting adaptability and achieving an acceptable level of accuracy, it does

not produce explainable results, cannot detect hidden dependencies, is not scalable with large requirements, and has generalizability issues.

Researchers [15] proposed an approach based on deep learning and NLP to extract implicit dependencies. It extracts cause-and-effect dependencies and then applies NLP-based methods to analyse the pattern that indicates these causal dependencies. It introduces an NLP-based architecture that uses neural networks and tree recursion. It is trained to discover the causality dependencies in human language. They also introduced a dataset that is suitable for deep learning. The results show the approach's ability to uncover causal relationships with acceptable accuracy and interpretable results. However, it cannot handle implicit dependencies. Researchers [16] proposed an RD detection framework that combines active learning with ontology-based and supervised ML. It applies active learning to refine models, reducing the manual effort for annotation. This approach is also ontology-based to benefit from specific domain knowledge and improve semantic requirements dependency analysis, affecting overall contextual relevance. Although it can detect implicit dependency and produce interpretable and accurate results, it has scalability concerns, as it has not been tested on a large dataset.

Researchers [17] introduced an optimization approach that combines human value dependencies with the RP process. This method represents RD via fuzzy logic, making modelling uncertainty flexible. This ensures high-value selection of the requirements while handling their dependencies via optimization methods. This paper highlights that the dependency representation can improve the decision-making process and enhance software release, and that fuzzy logic can achieve a realistic approach for complex requirement dependencies. This approach achieves high accuracy metrics in detecting implicit dependencies with a high level of explainability; however, it cannot handle large requirements and has generalizability. They also proposed an approach that combines fuzzy graphs with integer programming to control software dependencies [18]. This approach models each requirement as a node and each dependency among requirements as edges in the fuzzy graph, allowing flexible modelling of dependency uncertainties. It also balances technical dependency constraints, technical feasibility, and business value via integer programming, which optimizes release planning. The results highlight that handling dependency leads to more efficient software release planning. This approach can detect implicit dependencies and produce interpretable and accurate results. However, it has scalability concerns, as it has not been tested on a large dataset. This approach has been used as one of the baselines in this study because of its strong structure in dependency handling techniques.

Researchers [19] introduced an NLP-based technique called FSARC, which is short for the "Analysis-based Requirements Conflict Detector". It first analyses the finer semantic composition of the requirements to discover the conflict between them. The algorithm analyzes the linguistic features of the requirements and makes a notation for their semantic model construction. Although this technique can detect implicit dependencies with high accuracy, it has scalability, generalizability, and transparency issues. Researchers [20] introduced an LLM-based approach that discovers dependencies among requirements in complex systems. It uses LLMs to capture dependencies through contextual understanding. It integrates domain-specific information with the BERT model to improve the accuracy of dependency diversity. This approach performs requirement dependency classification and transforms the results into a list of all the dependencies among requirements. This approach can enhance consistency, traceability, and efficiency in controlling and managing dependencies among requirements in large software projects. However, it cannot handle implicit dependencies among requirements, has scalability concerns, has not been tested on a large dataset, and does not produce interpretable results.

Researchers [21] proposed a technique that uses supervised learning and semantic similarity to identify dependencies of type conflicts among requirements, namely, S3CDA. It has been applied in different domains where data are labelled, specifically in the software engineering field, which requires conflict identification. While applying contextual embedding, the method detects the semantic dependencies among requirements. The results of this approach depend highly on the quality of the labelled training dataset, which limits its adaptability to new scenarios. It also cannot handle implicit dependencies and cannot produce interpretable results. Researchers [22] have analysed the causal dependencies on software requirement prioritization. It utilizes NLP, including part-of-speech (POS), dependency parsing, and pattern-based matching, to discover causal relationships in the SRS. It also applies semantic similarity among words to extract causal dependency, reducing false positives. The extracted dependencies were explored further to assess their impact on requirements change management. The findings prove the importance of causality relationship detection in enhancing decision-making and avoiding risks. The approach in the study can handle implicit dependencies among requirements with acceptable accuracy. However, it does not produce explainable results and has scalability concerns, as it has not been tested on a large dataset.

Researchers [23] proposed a framework based on AI that enhances RD analysis from SRSs. It integrates ML with AL to minimize the complexity of dependency extraction on large labelled datasets. It applies ML to identify existing dependencies among requirements and adjusts them iteratively via active learning. It includes the domain experts' feedback to improve the model accuracy. The results show that it has achieved high accuracy metrics compared with traditional techniques, confirming the effectiveness of combining ML with AL in feedback loops. This approach can detect implicit dependency, but does not produce interpretable results and has scalability issues. Researchers [24] introduced an ML-based approach for dependency capture, focusing on analyzing their effects in real, practical scenarios. The approach integrates supervised and unsupervised learning models to discover complex dependencies from SRSs. It also uses ensemble learning, embedding, and

feature extraction to enhance the dependency extraction. They make a return on investment analysis to evaluate the trade-offs between computational cost and development complexity. The results clarify the benefits of applying ML models in RE while addressing domain-specific needs and scalability. This approach can detect implicit dependencies and has been tested on large datasets, but it has transparency issues.

Researchers [25] introduced a stacking ensemble learning model that extracts dependencies among requirements, namely, P-stacking. It applies semantic analysis via IDF features, POS, and Word2vec to enhance dependency prediction. It integrates various classification algorithms, such as neural networks, decision trees, and support vector machines. A meta-learner enhances the output of these models, producing a more accurate and dependency discovery mechanism. The model demonstrates high accuracy metrics compared with the traditional model, but it has not been tested on a large dataset and does not produce interpretable results. Researchers [26] introduced an LLM-based approach combined with a formal logic approach that extracts the dependencies of type contradictions between requirements. It focuses on identifying the conditional statements in human words. LLM helps translate complex, ambiguous, and context-dependent software requirements, enabling a deep understanding. Formal logic provides a structured framework for identifying contractions by interpreting the requirements set into logical rules. This hybrid approach emphasizes the benefits of applying LLM for human language translation and logic rules for structured validation. This work clarifies the method's ability to improve the consistency between requirements, specifically in large and complex projects. This approach can detect implicit dependencies with high accuracy. However, it has not been tested on large datasets and cannot produce interpretable results.

Researchers [27] introduced a framework, DepsRAG, that utilizes LLM models for managing dependency among requirements. This work uses retrieval-augmented generation (RAG) to enhance queries that retrieve relevant information from the resulting knowledge graph. It also applies a pretrained LLM model to analyse the resulting dependency graph. This proves improved accuracy in dependency extraction and the importance of including LLM models in RP tasks, specifically in discovering dependencies among requirements. It fills the gap between automated dependency extraction and decision-making in the RE field. This approach can detect implicit dependencies with acceptable accuracy; however, it has not been tested on a large dataset, and its results lack transparency. Researchers [28] introduced a static analysis-based approach, RefExpo, to extract dependency graphs from the requirements list. It focuses on the structural dependency type between requirements. It can extract many structures of large requirement sets via control flow analysis. This study demonstrates that automating the dependency analysis process enhances software maintainability and impact analysis by visualizing requirements dependencies. The resulting dependency graph provides a structured method for discovering strong dependencies, which achieves better decision-making. It outperforms manual approaches in terms of accuracy and efficiency. This approach produces interpretable results but cannot detect implicit dependencies and has scalability concerns. Researchers [29] introduced a technique that combines ML with NLP to extract conceptual diagrams from SRSs. This approach applies structured semantic analysis and representation to enhance the traceability and visualization of requirements. It helps to understand the dependencies among requirements by transforming the textual dependencies in the SRS into a conceptual model. It focuses mainly on discovering structural dependencies among requirements. Although the approach supports visualization and produces explainable results, it suffers from a scalability challenge and cannot detect implicit requirements efficiently.

Therefore, while existing RD analysis techniques provide valuable insights, most face transparency or scalability concerns, some lack evaluation on large real datasets, and others suffer from handling implicit RDs. Our proposed algorithm (DA-EAHP) overcomes these limitations by providing a more scalable, dependency-aware, transparent, robust, and comprehensive solution for RPs and RDs. Among these reviewed techniques, the fuzzy-logic-based approach [18] was selected as a strong comparative baseline in this paper because it focuses on "requires"-type RD, which is targeted in our research, and because of its strong structure in RD handling techniques. Table I compares each technique’s strengths, weaknesses, and key performance results.

TABLE I. A COMPARISON OF THE ADVANTAGES, LIMITATIONS, AND KEY PERFORMANCE INDICATORS FOR EACH TECHNIQUE.

Ref.	Year	Advantages	Limitations	key-performance
[14]	2020	• It handles the changing and evolving requirements. • It combines many phases, such as preprocessing and analysis, to improve the quality of the results.	• Results rely highly on the clarity of input. • Needs high computational resources. • Needs customization for each project. • Applicable only to small datasets. • Results are not interpretable.	• Evaluated on a real dataset and achieved an accuracy of 82% and a recall of 80%.
[15]	2020	• Offers a strong semantic mechanism for discovering cause and effect relationships and can handle complex dependencies. • Support handling scenarios that need detailed RD analysis. • Produce explainable results.	• Requires complete and structured input to produce high-quality results. • Needs customization for each different domain. • Need high computational resources. .	• Evaluated on a large and real dataset and achieved an F1 score of 81.5 %.

Ref.	Year	Advantages	Limitations	key-performance
[16]	2020	• Offer iterative learning, which optimizes time and resources. • Can handle semantic and logical dependencies. • Integrate AL with ontology-based dependency analysis.	• Needs well-structured ontologies for correct output. • Need customization for each domain. • Need high computational resources and involvement of domain experts. • Can handle only small datasets. • Results not explainable.	• Evaluated on real dataset and achieved F1 score of 86.3 %.
[17]	2020	• Offer optimization methods and provide a balance between the preferences of stakeholders and dependencies. • Handles RD uncertainties. • Handle human values dependencies. • Produce explainable results.	• Needs high computational resources due to the tuning of fuzzy parameters. • Can handle small datasets only. • Needs customization for each project.	• This approach cut value loss to about 5 %, while other methods lost 20-30%.
[18]	2021	• Offer a high level of transparency. • Handles uncertainty in RD • handles semantic dependencies. • Offer balancing of constraints and RD that support release planning.	• Needs high computational resources due to integer programming complexity. • Needs customization for each project. • Needs accurate modeling for determining the membership function.	• This method increased the total captured value and reduced the loss, specifically for large datasets.
[19]	2021	• Offer fine-grained dependency analysis to capture requirements conflicts. • Can identify the inconsistencies between requirements, reducing human effort. • Can handle hidden dependencies.	• Need high input quality to produce accurate results. • Need high computational resources. • Needs customization for each project. • Results are not explainable • Can handle only small datasets.	• Evaluated on two real datasets and achieved an F1 score of 84.3 and a Precision of 85%.
[20]	2022	• Can handle implicit and complex dependencies. • Can handle functional and structural dependencies. • Can be applied to different contexts and projects.	• Relied on LLMs, which introduce biases and subjectivity according to the training datasets. • Requires high computational processing. • Need high computational resources. • Can be applied only to small datasets. • Results lack explainability.	• Assessed on a real dataset and achieved an F1 score of 98 %.
[21]	2022	• Offers a supervised learning mechanism that improves the accuracy. • Handles synonyms and contextual variation and inconsistencies using semantic similarities. • Can be applied to different projects and contexts.	• Need highly complete and structured input to produce high-quality results • Needs annotated data to work, which is scarce in the software engineering field. • Can handle only explicit RD. • Can handle only small datasets.	• Assessed on a real data set and achieved a Recall of 85%, a Precision of 90 %, and an accuracy of 88 %.
[22]	2023	• Offer traceability by determining the causal links. • Can handle hidden dependencies. • Can be applied to different contexts.	• Need domain-specific fine-tuning. • Results lack explainability.	• Assessed on 2000 requirements, and produced an F1-score of 82 %.
[23]	2023	• Can handle structural and semantic dependencies. • Can handle new dependencies.	• Needs well-structured AL strategies. • Need customization for each project. • Can be applied only to small datasets • Results lack explainability.	• Assessed on a real dataset and produced an accuracy of 90%.
[24]	2023	• Offer cutting-edge mechanisms to enhance the dependency accuracy. • Offer complete analysis of costs and benefits using ML methods • Can handle implicit, semantic, structural, and evolving dependencies.	• Requires experts and high computational resources. • Need high computational resources. • Need involvement of domain experts. • Can handle only limited scenarios. • Results lack explainability.	• Assessed on various real datasets and improved the F1 score by 27%.
[25]	2024	• Supports integration of different ML models.	• Need high computational resources • Need correct parameter tuning and coordination between learners.	• Assessed on three real-world datasets and produced an F1 score of 90.2 %.

Ref.	Year	Advantages	Limitations	key-performance
		• Offer integration of various ML models • Can handle semantic and syntactic dependencies. • Can be applied to different contexts and projects.	• Can be applied only to small datasets. • Results lack explainability.	
[26]	2024	• Offer complex requirements dependency management • Handle hidden dependencies and nuanced contradictions. • Can be applied to large requirements	• Needs high computational resources because of the mapping of requirements into formal logic • LLMs introduce inaccuracies and biases.	• Assessed on a real dataset and produced 92.1% accuracy.
[27]	2024	• Can handle hidden and complex dependencies. • Produces explainable results.	• Need a large training dataset for producing accurate results • Need large computational complexity due to LLM model inference. • Need specific customization for each different project. • Can be applied only to small datasets.	• Assessed on a real data set and produced a Precision of 93 %. And an F1 score of 91.1%
[28]	2024	• Offer RD graphs and static analysis that enhance visualization analysis. • Produce explainable results.	• Needs regular updates to adapt to the changes of the programming language used, and is limited only to simple code. • It can be applied to only small datasets. • Cannot handle implicit dependencies.	• Assessed on a real dataset and achieved a precision of 92 % and a Recall of 88%
[29]	2025	• Enhance the requirements visualization and support traceability. • Reduce the requirements ambiguity. • Produce explainable results. • Applicable to different contexts.	• Can only handle explicit dependencies and static dependencies. • Can only be applied to small datasets.	• Tested on a real dataset and achieved an F1 score of 90%.

3. BACKGROUND

3.1 Requirement Dependencies (RDs)

RD plays a critical role in the RP process [6]. They can be categorized into two types: implicit dependencies and explicit dependencies. Explicit dependencies among requirements mean that the interdependence is obvious and can be extracted easily through basic dependency analysis techniques. On the other hand, implicit dependencies among requirement pairs are hidden and require stronger analysis and reasoning abilities for extraction. This type of dependency is considered a challenge, as its extraction requires strong semantic analysis and domain knowledge. Dependencies among requirements can also be classified as internal or external. Internal dependencies occur because of internal system attributes. It has many types, such as combination, exclusion, and implication (requires). Combination dependency is a pair of requirements that should be developed together. Exclusion refers to a pair of requirements that can be developed together. The implementation of a relationship is a requirement that needs to be implemented first (which is our focus in this paper). External dependency is the external attribute that affects the system. It has two common types: value-related, which are requirements that affect the cost or revenue of the system, and time-related, which means that some requirements must be implemented within a specific deadline and project schedule.

3.2 Large Language Models (LLMs)

In recent years, LLMs have achieved high performance in NLP tasks, such as contextual understanding, semantic similarity analysis, and RD extraction [23,26]. They have been considered among the most powerful tools in RE (especially in relationship analysis among requirements because of their superiority in capturing deep semantic meaning [23,26]. In the RE field, they are superior because of their ability to understand contextual meaning, which helps in requirements analysis. They introduce significant benefits in the RE field by automating manual tasks that require extensive human effort, such as requirement elicitation, prioritization, and classification. They also offer contextual understanding, which helps in dependency analysis and can streamline the overall requirements analysis, improving the efficiency and accuracy of the software development process. For dependency analysis, LLMs can be trained and fine-tuned on a set of requirements,

where each pair has no dependency or has a forewarned or revised dependency. This is helpful when requirements are written in human language, and LLMs help to understand the relationships among them.

LLMs are also integrated with a semantic similarity measure; for example, embeddings produced by large models can be calculated and measured via cosine similarity to detect coherent requirements. This approach reduces negative and positive false positives and enhances robustness. In our proposed algorithm, LLMs are the main stage in the dependency analysis mechanism after NLP analysis [23]. This first stage uses syntactic features and grammar to filter nonrelated pairs of requirements and detect related ones. This stage applies a fine-tuned model to the detected requirements pairs, performing an accurate dependency analysis via its deep domain and contextual understanding. This hybrid of NLP and LLM balances precision and efficiency: NLP reduces the search space by detecting grammatical cues, and LLM ensures high-quality dependency analysis. Overall, LLMs are a transformative advancement in requirements dependency analysis. Their adaptability to fine-tuning, ability for contextual reasoning, and integration with a similarity-based approach make them robust and powerful components in a new dependency-aware prioritization framework. Several studies have emphasized that LLM models can outperform traditional keyword and rule-based techniques in discovering complex and implicit RDs that are not obvious or explicitly stated [12,20,25,26]. They consider them one of the state-of-the-art baselines for semantic extraction in modern RE workflows. Therefore, this study considered LLM models as one of the strongest baselines to assess the efficiency and effectiveness of our proposed dependency-aware RP algorithm.

3.3 Enhanced-Analytical Hierarchy Process (E-AHP)

Many researchers emphasize that the AHP [30] is one of the most accurate RP techniques, as it applies strong mathematical equations [31-34]. However, the AHP is suitable only when a small number of requirements are prioritized, as it is based on pairwise comparison, which increases exponentially with the number of requirements [13,31]. Like the other RP techniques, it also cannot handle RDs. Furthermore, the results become inconsistent with many requirements.
Therefore, the AHP has four main limitations [13,31]:
So, the AHP has four main limitations [13,31]:

1. Its performance degrades significantly with large requirement numbers.
2. It produces inconsistent results as its steps include a high level of human involvement.
3. It is not easy to understand how to use, especially the consistency checking mechanism
4. It does not handle RD during the RP process (like almost all RP techniques).

The first, second, and third issues have been addressed in our previously published research by introducing the E-AHP algorithm [13], which enhances the scalability and results consistency of the AHP, enabling it to handle large requirements with acceptable accuracy. E-AHP [13] introduces algorithmic refinements that significantly reduce computational complexity (time and memory consumption) and enhance the consistency of the results. The E-AHP solves the AHP problems as follows:

- It increases the scalability by minimizing the time the AHP takes in the pairwise comparisons. It groups the coherent requirements, making pairwise comparisons among groups of requirements instead of separate requirements, which decreases the matrix size, saving time and memory.
- It reduces the inconsistency of the AHP by allowing users to score each group of requirements instead of performing pairwise comparisons among each pair of requirements.
- It also introduces another new consistency checking mechanism instead of the complex mechanism in the AHP, making it simple and easy to understand.

E-AHP was validated via a large dataset (up to 500 requirements) against conventional AHP and another recent improvement algorithm called ReDCCahp [35]. It has proven superior performance in terms of scalability, RP accuracy, and result consistency [13]. However, it does not address the fourth limitation, as it does not consider RD during the RP process. This research refines our previous E-AHP algorithm by integrating a novel AI-based dependency-aware mechanism into the RP process, enhancing its realism and accuracy in real-world and complex software projects. Moreover, we refine the requirements scoring and grouping mechanisms of E-AHP by replacing heuristic rules with structured rules. (Fig.1). shows the differences and improvements in E-AHP over AHP, and DA-EAHP over E-AHP.

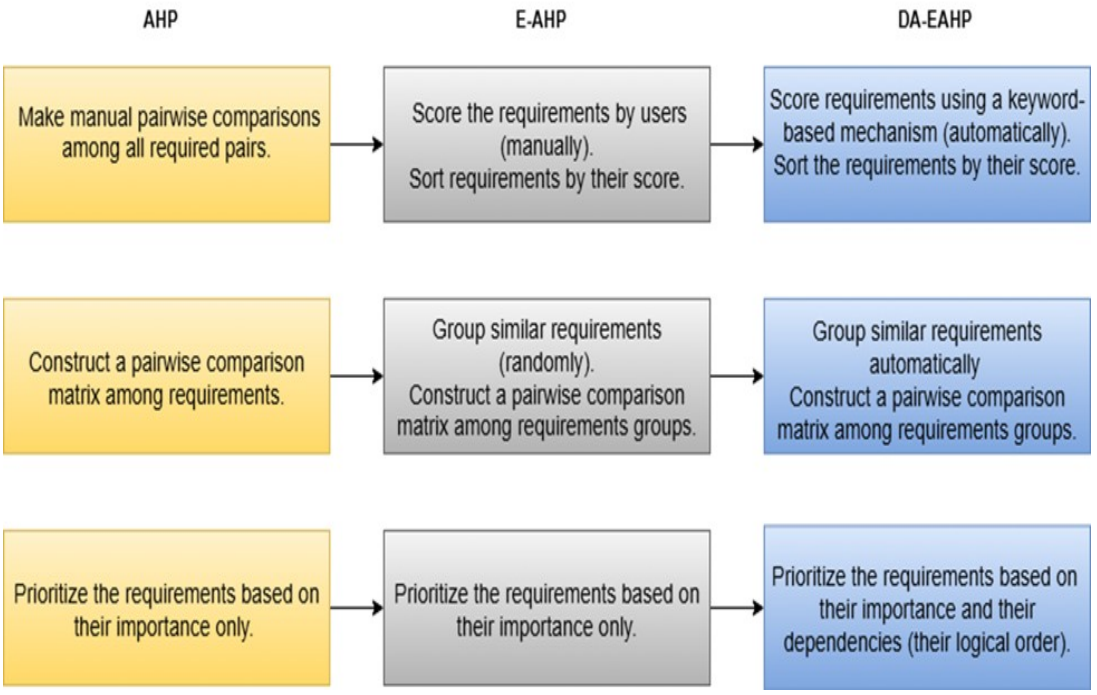

Fig.1. The differences and improvements in E-AHP over AHP, and DA-EAHP over E-AHP

4. PROPOSED ALGORITHM: DA-EAHP

This section presents our proposed algorithm, DA-EAHP, an enhanced version of our previously proposed E-AHP algorithm [13]. While the E-AHP focuses on prioritizing large requirements, the DA-EAHP improves it further by considering RD during prioritization, specifically of the type “requires. This enables more accurate and context-aware prioritization results that reflect real-world constraints among software requirements. The following subsections describe the algorithm steps in detail.

4.1 Requirements Gathering and Labelling

The first step is to gather and specify all the requirements that will form the SRS, input for the analysis phase. Each statement will represent one requirement. The analysts should ensure that the requirements list in the SRS is complete, clear, and consistent. Each requirement will have an ID, making it easy to modify and refer to. These IDs will have a format like R1, R2, R3, etc. Mapping is built to connect each requirement to its ID and vice versa. These mappings are critical for efficiently referencing and processing the requirements in the next phases.

4.2 Keyword-based requirements Scoring

In this step, we implement a scoring system to assign a score to each word in each requirement statement. Some important fixed and dynamic keywords are defined and weighted (scored) based on their importance within the project context. Fixed keyword scoring means defining common keywords in most projects, such as ‘login’, and giving them a specific score. Dynamic keyword scoring assigns a score based on the ability to extract project-related keywords dynamically by analyzing the context and frequency of the terms in the SRS document. In this step, a final score for each requirement statement is calculated by summing the score weights for each word in the statement based on the dynamic and fixed keyword weights. After the scoring process is complete, the algorithm sorts the requirements by their scores in descending order. A hybrid algorithm of merge sort and insertion sort [36] is applied to sort the list of requirement scores.

4.3 Automatic Grouping for Coherent Requirements

The algorithm in this phase groups requirements whose score difference is less than the maximum difference requirements score (MaxDRS), where MaxDRS represents the allowed score difference among the requirements in a group. Another defined variable is the maximum number of requirements (MaxNR), which is the maximum number in the same group. The algorithm automatically calculates MaxDRS via the following equation.

$$MaxDRS = \frac{\sigma(S(R_i))}{2} \quad (1)$$

where σ is the standard deviation (SD) of the score values. The SD measures the dispersion or spreads of the score values around the mean [37]. The grouping threshold is set to half MaxDRS to balance specificity and sensitivity. Dividing it by 2 enables group variability and prevents grouping the requirements with highly divergent scores, ensuring that only the requirements with sufficiently strong dependency or semantic links are grouped [13]. For example, if the SD is 4, then MaxDRS will be 2, which means that the difference in the requirements in the same group is less than or equal to 2. It adapts to the data characteristics and ensures that the requirements are grouped based on the spread of scores. The algorithm calculates the value of MaxNR via the following equation:

$$\text{MaxNR} = \sqrt{\text{int}}(n) \quad (2)$$

Where n is the number of requirements, and the square root is used, as it offers a balanced way to specify the group size. It scales appropriately with the dataset size, as the group sizes are neither too small nor too large. Therefore, when the square root function is applied to the requirements grouping process, it adjusts automatically based on the total number. For example, if the number of requirements is 25, MaxNR will be 5, which means that the maximum number of requirements in one group is 5. To assign a general score for each group, the algorithm calculates the average score of all included requirements in that group.

4.4 Constructing the pairwise comparison matrix

The algorithm then creates a pairwise comparison matrix for each group of requirements. The columns and rows of each matrix represent the requirement groups, and the matrix cells are the score differences among the groups. For example, if the score of Group 1 (G1) is 4 and the score of Group 2 (G2) is 2, then the value of the intersection cell between G1 and G2 will be 2 (4/2) and will be .5 (2/4) between rows G2 and G1. The algorithm then normalizes the matrix to ensure that each column has a sum of one. After that, the algorithm performs the same mathematical operation as the AHP to obtain the initial PV.

4.5 Dependency Analysis Process

In alignment with the RP process, the algorithm includes an RD extraction mechanism that focuses on the "requires" dependency type among requirements, effectively ensuring the implementation of the requirements in the correct logical order. This makes the RP not only based on important aspects but also on the functional RD. To achieve effective dependency analysis, the process consists of two layers that combine the power of both the LLMs and the NLP, which enables the extraction of two main types of RDs: explicit and implicit. The extracted RD from the following two RD handling steps is combined into a one-directed graph, where each node represents a requirements statement and each edge represents a required-type relationship between them [18].

4.5.1 Syntactic Dependency Extraction (via Rule-Based approach and NLP)

The step applies a widely used NLP library called “SpaCy” [38] to perform lexical and syntactic analysis on each requirement statement. It includes common preprocessing steps such as tokenization, part-of-speech tagging, and dependency parsing. It serves as an interpretable mechanism for capturing grammatically implied or explicit dependency, as it focuses on structurally apparent cues and provides an explainable and robust foundation upon which the semantic dependency analysis can be built. The algorithm applies heuristic rules to discover grammatically supported and explicit RDs. These rules include the following: (1) Keyword-based dependencies, which are identified via the presence of sequencing words such as “required”, “after”, “dependent on”, and “must”. (2) Referential links: where the determiners or pronouns such as “it”, “that”, and “this”, refer to requirements that were mentioned previously, showing contextual dependency.

4.5.2 Semantic Dependency Extraction Using LLMs

This stage combines the LLM's semantic similarity computation to uncover non-explicit RDs and conceptually meaningful relationships that may not be discovered via keywords or grammatical rules in the previous step. We specifically fine-tuned a pretrained bidirectional encoder representation from the Transformers (BERT) model [39] from Hugging Face [40] to determine "requires"-type RD. BERT was selected because it has been proven effective in semantic analysis and RD handling in prior studies [20,25,39,40]. Moreover, it can be fine-tuned in domain-specific tasks, allowing our algorithm to detect dependencies accurately and enhance performance while maintaining computational and memory consumption efficiency. First, it encodes each requirement statement into a semantic vector representing the contextual meaning of requirements, enabling deeper semantic understanding. Then, a cosine similarity is calculated among each pair of vectors; a similarity is indicated when the similarity score among vectors exceeds a predefined threshold (for example, cosine similarity is above .8). This step identifies explicit relationships, such as “setup” before “use” and “authentication” before

"access". For example, if there are two statements: "The system should enable the users to sign in securely" and "The user can reset the password ", this pair of requirements has no explicit relationship terms; their semantic similarity indicates a prerequisite dependency.

4.6 Integration of dependency analysis with prioritization results.

The priority score in the PV of each requirement is adjusted based on the resulting RD to reflect the correct logical implementation order. The dependent requirements will have lower priority, and the autonomous requirements will have higher priority. The algorithm considers direct and transitive RD. If a requirement directly depends on another requirement, its priority must reflect the importance of all prerequisite requirements. If one requires another indirectly, its priority must account for all preceding requirements. The algorithm then updates the criteria score in the priority vector via the following equation:

$$PV'[i] = PV'[j] * (1 - DIF) + \varepsilon \quad (3)$$

The DIF variable refers to the dependency influence factor, which means the highest number of dependencies for a specific requirement in the RD sets. When the priorities are updated, the priority score is updated with the resulting DIF, and the algorithm normalizes the scores again to ensure that their summation is 1. Note that to prevent result distortion due to zero values and maintain a fair and balanced contribution of all requirements (if the PV calculation is repeated later, dependency relationships have been updated or changed), a constant (.001) is added to each new PV to ensure numerical stability and preserve the proportional relationship. After that, the algorithm will sort the requirements in descending order by their new updated score and output the final priority list. For example, let us consider the values in Table II. Assume that there are 5 requirements: R1, R2, R3, R4, and R5. Their score values in the PV after matrix construction are (0.15, 0.25, 0.2, 0.3, 0.1). Fig. 2 shows the RD values in Table II.

TABLE II. EXAMPLE OF REQUIREMENTS AND THEIR DEPENDENCIES.

Requirements	Depends on	# Dependencies
R1	R3	1
R2	R1, R3	2
R3	-	0
R4	R1, R2, R3	3
R5	-	0

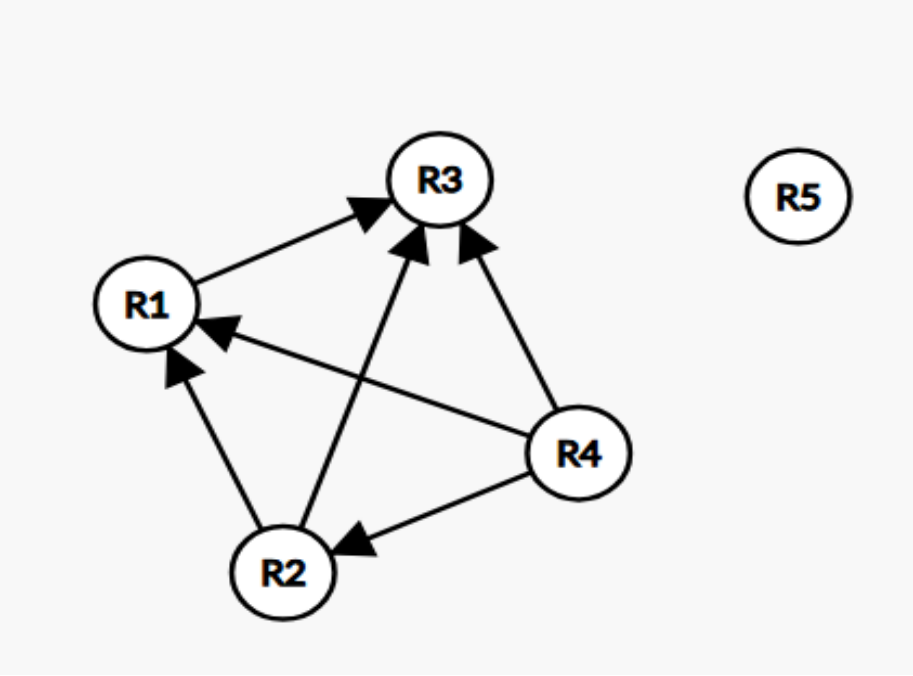

Fig.2. Example of Requirements and Their Dependencies.

In this example, the maximum dependencies are 3 (as R1, R4, and R2 depend on R3), so the DIF here is 3. By applying the following equation, we can obtain the updated normalized priority vector score for requirements and their normalization (by dividing each updated PV value by its new sum) via the following equation:

$$PV'[i] = \frac{PV'[i]}{\sum PV'[j]} \quad (4)$$

Equations (3) and (4) can be integrated into the following equation:

$$PV''[i] = \frac{PV'[i] * (1 - DIF)}{\sum PV'[j] * (1 - DIF\ ) + \varepsilon} \quad (5)$$

So, by compensation in (3) and (4), the final updated priority values are as follows:

$PV'(R_1) = \left(1 - (\frac{1}{3})\right) * .15 + .001 = 0.10$	$PV''(R_1) = \left(\frac{.1}{.48}\right) = 0.20$
$PV'(R_2) = \left(1 - (\frac{2}{3})\right) * .25 + .001 = 0.08$	$PV''(R_2) = \left(\frac{.08}{.48}\right) = 0.16$
$PV'(R_3) = \left(1 - (\frac{0}{3})\right) * .20 + .001 = 0.20$	$PV''(R_3) = \left(\frac{.2}{.48}\right) = 0.41$
$PV'(R_4) = \left(1 - (\frac{3}{3})\right) * .30 + .001 = 0.001$	$PV''(R_4) = \left(\frac{.0}{.48}\right) = 0.001$
$PV'(R_5) = \left(1 - \left(\frac{0}{3}\right)\right) * .10 + .001 = 0.10$	$PV''(R_5) = \left(\frac{.1}{.48}\right) = 0.20$

So, we can note here that the PV for R3 has been increased from 0.2 to 0.41, as many requirements depend on it (R1, R2, and R4); on the other hand, the PV of R4 has been decreased from 0.3 to 0.001, as it depends on many requirements (R1, R2, and R3). We can compensate directly in equation (5), but we chose compensation in (3) and (4) in the example to increase clarity. Note that if multiple requirements are the same, PV'', such as R1 and R5, the algorithm will keep the original PV value from step 4.4. Therefore, R1 will have a higher rank than R5 as it has a higher PV value. The flow chart in Fig.3 and Algorithm 1 explain and clarify our proposed algorithm:

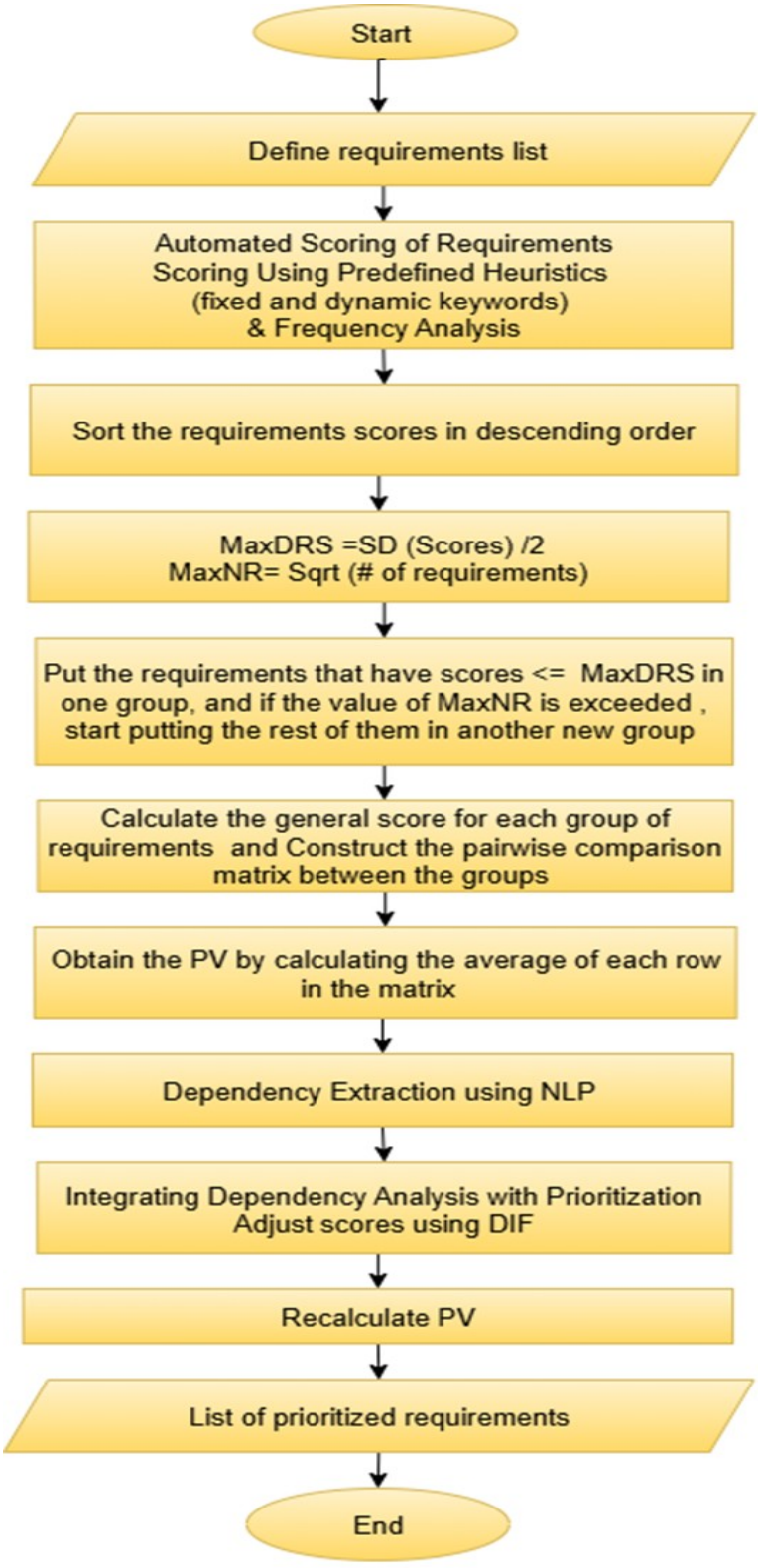

Fig.3. Flow Chart for The Proposed Algorithm.

Algorithm 1: Dependency-Aware E-AHP (DA-EAHP) Algorithm

Input data: Requirements list $R = \{R_1, R_2, \ldots, R_N\}$ where N is the total number of requirements.

Output: Prioritized-dependency-aware requirements list R'.

Step 1: Score requirements in R as follows:

Step 1.1: Specify a list of fixed keywords K_{fixed} with an assigned fixed weight W_{fixed} for each K_{fixed}.

Step 1.2: Extract dynamic keywords $K_{dynamic}$ Using context and frequency analysis, assign a dynamic weight$W_{dynamic}$ for each $K_{dynamic}$.

Step 1.3: for each R_i, calculate score $S_{fixed}(R_i)$, as the sum of W_{fixed} for each K_{fixed} in R_i.

Step 1.4: for each R_i, calculate score $S_{dynamic}(R_i)$ as the sum of $W_{dynamic}$ for each K in R_i.

Step 1.5: for each R_i, calculate the total score $S(R_i) = S_{fixed}(R_i) + S_{dynamic}(R_i)$.

Step 2: Sort requirements R by their scores $S(R_i)$ using a hybrid algorithm of merge sort and insertion sort algorithms.

Step 3: Calculate *MaxDRS* using Eq. (1), and *MaxNR* using Eq. (2), group sorted requirements while ensuring that the difference between all scores in one group is ≤ *MaxDRS*, and the number of requirements per group is ≤ *MaxNR*.

Step 4: for each requirement group pair G_k and G_j, where k is the row and j is the column of the matrix M, create a PWC Matrix M_{kj} based on their score values, if $S(G_k) = S(G_j)$, Set $M_{kj} = 1$, else set $M_{kj} = \frac{S(G_k)}{S(G_j)}$.

Step 5: Normalize M by dividing each cell by the sum of its column as follows $M'_{kj} = \frac{M_{kj}}{\sum M_j}$.

Step 6: Calculate the PV for each G_k by summing its row in M' and dividing it by N. $PV(G_K) = \frac{M'_k}{N}$.

Step 7: Extract the existing RD:

Step 7.1: Parse each requirement R_i using SpaCy to identify explicit RD.

Step 7.2: Each pair of requirements (R_i, R_j) is compared semantically by generating semantic embeddings using a fine-tuned pre-trained BERT model to detect whether a "requires" dependency exists between R_i and R_j

Step 7.3: Assuming *D* is the dependency list

If similarity (Ri, R_j) ≥ 0.8, then:

If Ri depends on R_j, then:

Then, add ($R_i \rightarrow R_j$) to *D*.

If R_j depends on Ri, then:

Then, add ($R_j \rightarrow$ Ri) to *D*.

Step 7.4: Construct a graph G Nodes represent requirements, and edges represent dependency relationships among them.

Step 8: Integrate Dependency extraction with prioritization results:

Step 8.1: Update the PV using DIF by applying Eq. (3) to get PV'.

Step 8.2: Normalize the updated PV, making $\sum PV'_i = 1$ by applying Eq. (4) and getting the final normalized priority vector PV'', or get it directly by applying Eq. (5) instead of (3) and (4).

Step 9: Sort requirements based on PV'' to get R'.

5. EXPERIMENTAL SETUP

This section discusses the research experimental details that compare our proposed algorithm against two recent RD handling algorithms in terms of accuracy and performance (computational complexity), with variant data set sizes ranging from small to large. The following subsections mention the experiment objectives and set-up details.

5.1 Experiment Objectives

The objective of the experiment is to assess our novel proposed dependency-aware DA-EAHP algorithm, a fuzzy-based model[18], a zero-shot GPT-3.5 LLM-based model, and a zero-shot robustly optimized BERT approach (RoBERTa), based on their RD analysis accuracy, computational complexity, and scalability (note that we don't evaluate the prioritization results, as our previous version of our algorithm, E-AHP, has already achieved superior performance in RP, and also, there are no current existing algorithms that integrate both process handling and RP to compare with). Below are the research questions of our research:

RQ1: What is the performance of the three algorithms concerning completion time and memory usage when dealing with different data set sizes?
RQ2: What is the accuracy of the three algorithms' dependency extraction when dealing with different data set sizes?
RQ3: What is the impact of incorporating the dependency awareness, specifically of type "requires", on the RP process?

5.2 Experiment Setup

We compare three algorithms in this paper: (1) our proposed algorithm, DA-EAHP, which integrates the RD analysis process into E-AHP; (2) a recent RD handling model [18] that integrates a fuzzy graph with integer programming, which is selected because it focuses on extracting RD of type "requires", which aligns with our focus and has a strong structure; and (3) a purely LLM-based model that uses the GPT-3.5 model in the zero-shot setting, without any fine tuning (4), and the zero-shot RoBERTa model [41], all of which were evaluated on the PURE dataset, due to its popularity in the RE field [42]. These baselines enable a fair comparison among the integration of rule-based and fine-tuned BERT LLM models, fuzzy models, and purely data-driven LLM models in zero-shot settings. These algorithms were evaluated in terms of two main criteria: accuracy and performance.

The accuracy assessment evaluates each algorithm's RD extraction results via different accuracy metrics, including the recall, F1-score, and precision. To create the ground truth for RD assessments (because the PURE dataset and most of the RE real-world datasets lack RD labels) [42], five domain experts (senior software engineers, selected based on their strong background and practical experience; exceeding 8 years; in the software engineering field.) were asked to extract the 'require' dependency type between each pair of requirements from 50 randomly selected requirements from the PURE dataset, which builds a solid ground truth and reference to evaluate the RD extraction accuracy of our algorithms (the use of 5 experts was sufficient, as the AHP relies on subjective judgment rather than a large number of requirements [30]). Then, a Google form containing the results of each algorithm (the detected pair of RDs) based on the same requirements was given to the experts to evaluate their accuracy by comparing the three algorithms' dependency extraction results with the expert results.

To conduct a performance and scalability assessment for the three algorithms, they were evaluated on various-sized subsets from the PURE dataset, ranging from 25 500 requirements, which are small, medium, and large datasets. The dataset was divided into three subsets: 80% for the training task, 10% for the validation task, and 10% for the testing task. Each dataset sample is saved as a text file, with one requirement per line, keeping the structure and meaning consistent across all datasets. The execution time was calculated and measured in seconds (sec). Memory usage was also calculated and measured in megabytes (MBs).To assess the impact of adding a dependency awareness mechanism in our proposed RP algorithm, we compare it with another variant that neglects the RD analysis phase. We ask the experts to prioritize the same 50 requirements based on their logical order. The form contains the prioritization results for the requirements of the two versions, which are assessed via top-k agreement to compare the top-k requirements resulting from each version (with and without RD handling) against the prioritization list produced by experts. It reflects the number of top-k-highest requirements correctly identified by each version, taking the experts' results as the ground truth.

The experiment was conducted on a machine with a CPU: 11th Gen Intel(R) Core (TM) i7-11800H @ 2.30 GHz, 2.30 GHz, RAM: 16.0 GB, system type: 64-bit operating system, x64-based processor, and OS: Windows 11 version 23H2. The three algorithms were written in Python version 3.11. The (en_core_web_sm) library from the SpaCy model was used for keyword parsing and extraction [38]. For RD extraction and semantic similarity, (all-MiniLM-L6-v2) from sentence-BERT was used [39]. The Matplotlib library is used for data visualization, the NetworkX library is used for graph analysis, Pandas is used, and the NumPy library is used for data manipulation [38]. To optimize the performance of our proposed algorithm, parallel processing is employed during the dependency detection phase, specifically for the pairwise similarity calculation among requirements, which is based on LLM embedding. The algorithm parallelizes the process across multiple CPU cores. This optimization reduces the time and memory requirements, making the algorithm more applicable to large real-world datasets.

6. RESULTS AND DISCUSSION

6.1 Results

This section evaluates the algorithms' accuracy and performance. Tables III and IV present the average time taken and memory used. The charts in Figs. 4 and 5 visualize the results in Tables III and IV, respectively. They show that our proposed algorithm demonstrates superior time and memory efficiency. The difference among the algorithms becomes large when the requirements increase, which proves our algorithm's scalability and performance efficiency.

Table III. Time taken by the DA-EAHP, zero-shot (GPT 3.5 and RoBERTa) LLM models, and fuzzy logic in sec.

Number of Requirements	DA-EAHP	Zero-shot GPT-3.5	Fuzzy logic	Zero-shot RoBERTa
25	3.4	9.9	18.5	11.6
50	5.7	14.2	34.3	17.4
100	13.2	27.1	68.5	31.5
200	25.8	53.2	146.8	62.8
300	38.7	81.4	223.7	98.5
400	52.9	107.2	296.3	132.1
500	65.2	136.7	361.2	168.0

TABLE IV. MEMORY CONSUMPTION BY THE DA-EAHP, ZERO-SHOT (GPT 3.5 AND ROBERTA) LLM MODELS, AND FUZZY LOGIC IN MB.

Number of Requirements	DA-EAHP	Zero-shot GPT-3.5	Fuzzy logic	Zero-shot RoBERTa
25	67.2	204.7	114.2	221
50	89.9	281.2	161.5	301
100	153.2	522.1	363.9	584
200	268.2	971.2	784.6	1131
300	372.3	1438.2	1189.9	1675
400	481.8	1901.7	1581.5	2234
500	590.5	2401.3	2052.7	2752

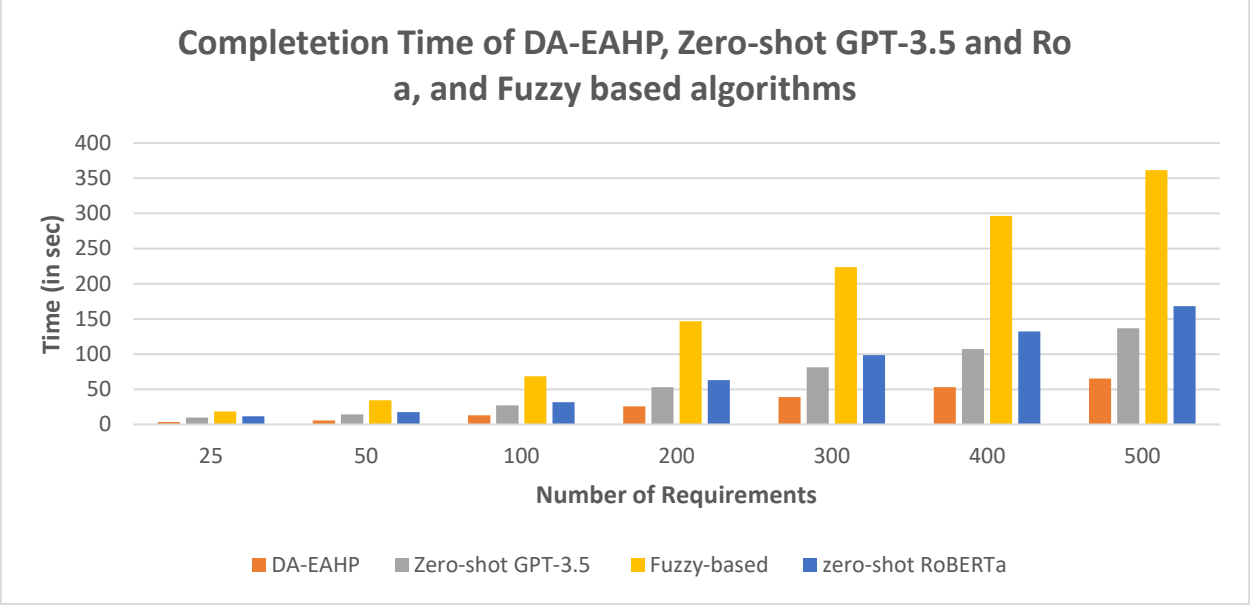

Fig.4. Time taken by the DA-EAHP, DA-EAHP, and zero-shot (GPT 3.5 and RoBERTa) LLM models, and the fuzzy logic zero-model in sec.

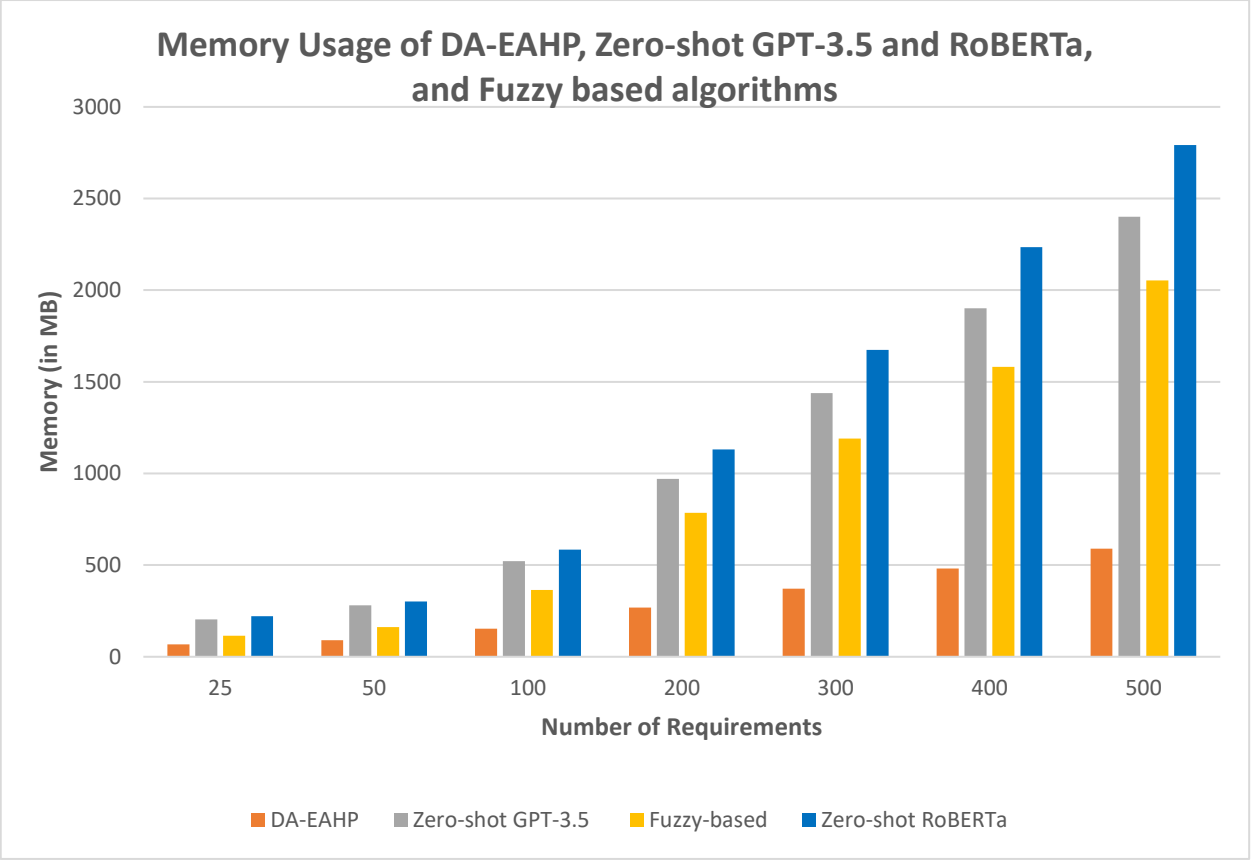

Fig.5.Memory consumption by DA-EAHP, zero-shot (GPT 3.5 and RoBERTa) LLM models, fuzzy logic in MB

Table V shows the accuracy, recall, precision, and F1-score of the proposed method based on the experts' annotations for RD. Our proposed algorithm outperforms the zero-shot LLM and fuzzy graph-based models, aligning with the judgments of the experts involved, as it achieves the highest accuracy metrics based on the experts' annotated dependencies.

TABLE V. ACCURACY METRICS FOR DA-EAHP, ZERO-SHOT GPT-3.5, FUZZY-BASED, AND ZERO-SHOT ROBERTA

Accuracy Metric	DA-EAHP	Zero-shot GPT-3.5	Fuzzy-Based	Zero-shot RoBERTa
Accuracy	0.92	0.85	0.736	0.82
Precision	0.92	0.84	0.72	0.81
Recall	0.93	0.86	0.75	0.80
F1-Score	0.90	0.85	0.73	0.816

We perform a paired t-test between DA-EAHP and each baseline mode to assess the significance of the performance differences. Specifically, DA-EAHP achieved a 6.75%enhancement (SD = 1.8%) over zero-shot GPT 3.5 ($t = 10.7$, p-value = 0.0017), a 10.6% enhancement (SD = 2.1%) over the zero-shot RoBERTa model ($t = 11.01$, p- value = 0.0016) and an 18.35% enhancement (SD = 2.4%) over the fuzzy model ($t = 29.42$, $p < 0.0001$). Therefore, the results show that DA-EAHP significantly outperforms the other baselines, with p-values <0.001. These results confirm the statistical significance and robustness of our proposed algorithm's superiority. Table VI shows the top-k agreement of the prioritization results produced by the two versions of the DA-EAHP algorithm (with and without dependency) with the experts' prioritization results. It shows the proportion of top requirements in terms of agreement between the results produced by each version and the experts' results. For example, if the tested algorithm and expert produce the same top 8 highest requirements out of the top 10 requirements, the result in the table will be .8 (80%). It clarifies the impact of handling RDs in the RP process on prioritization accuracy based on agreements with experts. Table VI shows that handling RDs in the RP process impacts the RP positively.

TABLE VI. TOP-K AGREEMENT BETWEEN DA-EAHP (WITH RD HANDLING) AND EAHP (WITHOUT RD HANDLING) WITH EXPERTS' RESULTS.

Number of Requirements	DA-EAHP (With RD handling)	EAHP (Without RD handling)
10	0.90	0.80
25	0.88	0.79
50	0.87	0.73

6.1 Discussion

6.2.1 Performance evaluation

Figs. 4 and 5 show that our proposed algorithm has proven performance superiority in terms of time and memory. This happens for many reasons. One of the main reasons for complexity efficiency is the partial use of LLMs for computing the semantic similarity in RD detection instead of entirely depending on LLMs. It applies the LLM model only when semantic ambiguity among requirements is detected using lightweight NLP in an early stage. This selective use of LLM results in good semantic and context understanding while reducing computational overhead. Moreover, the fine-tuning of the BERT model minimizes computational complexity as the model learns through the training phase and no longer needs long runtime reasoning, making inference more efficient, lighter, and faster for repeated use. On the other hand, zero-shot LLM models, such as GPT-3 and RoBERTa, require considerable computational complexity because they rely on on-the-fly reasoning, making inference time usage and memory less efficient for repeated or large tasks. Another main reason is that the proposed algorithm avoids the classical large number of pairwise comparisons in the basic AHP (which increases exponentially with the number of requirements) that increases time and memory (as the matrix size increases and the number of operations increases); instead, it scores the requirements via a combination of dynamic and static key-weighted scoring mechanisms and applies early automatic grouping for the requirements based on their scores, which avoids encoding and comparing every pair of requirements, minimizes the matrix size, and the number of operations needed. These automatic scoring and grouping mechanisms lead to fast processing and a noticeable reduction in time and memory, achieving more scalable performance in practice.

On the other hand, the fuzzy graph-based model becomes computationally expensive, especially with large datasets, as it integrates fuzzy graphs with integer programming, which involves matrix operations and optimization. Like basic AHP, the size of matrix operations increases significantly with the requirements, which increases the computational overhead, including time and memory. Moreover, when the dataset size increases, the graph produced by the fuzzy-based model becomes unreadable and impractical to interpret, limiting its scalability, explainability, and usability in large systems. Furthermore, its algorithm relies on Elles' measure of strength [18], requiring triple-nested iterations among each requirement pair, which results in cubic time complexity. The algorithm also stores RD matrices and propagates their strengths through transactional relationships [18], resulting in high memory consumption and significantly reduced performance as the requirements increase.

6.2.2 Accuracy evaluation

From the RD extraction accuracy perspective, Table V shows that the proposed algorithm achieves the highest accuracy metrics compared with the other algorithms. The dual use of LLM and advanced NLP embedding enables the identification of explicit and implicit, nuanced, and hidden RDs, achieving higher accuracy metrics than when the LLM model is used only. The participants confirmed that the proposed algorithm outperformed zero-shot LLM models, such as GPT-3 and RoBERTa, and fuzzy-based models in detecting fine-grained relationships. On the other hand, the fuzzy-based model fails to capture implicit RDs, as it employs a fuzzy graph-based method to extract RDs, which relies solely on fuzzy thresholds and a prespecified relationship, thereby degrading its accuracy when dealing with subtle, new, or complex RDs that are not prespecified. Additionally, they may neglect semantic similarities and fail to capture RDs with the same meaning but different wording. This results in a higher rate of false negatives, which affects the accuracy of RD extraction. Another main reason for the superiority of the DA-EAHP model over the GPT-3 and RoBERTa models is that the pure-LLM models were assisted in a zero-shot setting. In contrast, DA-EAHP was designed and optimized for the target dataset. The tuning for the task enables DA-EAHP to achieve higher accuracy in RD detection, as it allows more context-aware and precise prediction, leading to more reliable output. Moreover, while the zero-shot LLM models work as a black box [43], the proposed algorithm has strong and clear structured steps, domain-specific keyword-based scoring, and a logic-based grouping mechanism. This provides clear transparency for the detected RD and prioritized ranking. This generally improves the prioritization results and decision transparency. DA-EAHP improves the k-agreements because it prioritizes the requirements in the correct logical

order, increases the prioritization accuracy, and enhances consensus among users. Therefore, the results show that it can handle large numbers and maintain accuracy, confirming its scalability and generalizability, as it achieves the best performance across different dataset sizes, showing robustness in small, medium-, and large-scale prioritization tasks. Overall, the proposed algorithm provides an effective balance between transparency, accuracy, computational complexity, and scalability, which is essential for a real-world software project.

6.2.3 Key findings

Below is a list of the summarized key findings:

- The DA-EAHP algorithm has less computational complexity than the baseline zero-shot LLM models due to its partial use of LLM semantics, in addition to lightweight NLP, instead of fully relying on LLM in the RD extraction phase.
- DA-EAHP outperforms baseline zero-shot LLM models in performance (computational complexity) and accuracy to its specific task design; on the other hand, it was evaluated in a zero-shot setting without fine-tuning.
- Although the baseline zero-shot LLM models produce accurate results [44] [45], they consume high amounts of memory due to the full attention given to all the tokens (input) and embedding generation, especially with large numbers, which affects their scalability and performance.
- The DA-EAHP algorithm provides high-level transparency and interpretability, as it has clear formal and structural rules. Unlike the zero-shot LLM models, which lack transparency, they function as black boxes. Additionally, in a fuzzy-based model, the graph becomes non-interpretable with a medium or large number of requirements, affecting its interpretability.
- DA-EAHP outperforms the baseline fuzzy-based model in performance, as the fuzzy-based model is based on fuzzy graphs, which become computationally expensive and time and memory-intensive with large requirements because it relies on pairwise matrix operations and optimization, making it scale poorly with the requirements. On the other hand, applying automatic and semantic scoring and grouping mechanisms into coherent requirements in DA-EAHP reduces time, memory, and human error.
- The baseline fuzzy-based model fails to capture implicit RD effectively, as it neglects semantic similarity and depends on only the fuzzy threshold and pre-specified relationships, reducing its accuracy when dealing with subtle, new, or complex RDs. On the other hand, the DA-EAHP and zero-shot LLM models can discover explicit and implicit RDs, making them suitable for modern systems with complex RDs.
- DA-EAHP increased K-agreements among domain experts, demonstrating its effectiveness in handling RDRDs and enhancing the prioritization results.
- Therefore, we can conclude that, compared with the other baseline models, the DA-EAHP algorithm achieves the best accuracy, performance, transparency, and scalability.

7. LIMITATIONS AND THREATS TO VALIDITY

Despite extensive experiments, many potential threats to validity should be acknowledged. For internal validity, subjectivity is introduced due to the expert's annotations in establishing the ground truth, as the research relies on expert-labelled ground truth, and the results can be sensitive to configuration parameters across the proposed algorithm scoring and LLM thresholds. The results may also depend on the requirements structure and phrasing. For external validity, the results rely highly on the selected LLM models; different models may output different results. The algorithm's scalability up to 500 requirements has not been tested. The proposed algorithm's performance might not be generalizable across all domains, as it was assessed only on the PURE dataset.

8. CONCLUSION

This research introduces a novel, comprehensive dependency-aware RP algorithm called DA-EAHP, which improves our previously proposed algorithm, E-AHP, by incorporating multilayer, nuanced semantic RD detection using LLMs and NLP into the optimized RP flow, which solves the challenge of neglecting RD while prioritizing. The experimental results show that our proposed algorithm demonstrates superior accuracy and performance even with large requirement numbers and shows its effectiveness in improving the prioritization results. It achieves the highest accuracy in RD analysis while maintaining the lowest computational complexity, significantly enhancing the RP process. On the other hand, although baseline zero-shot LLM-based and fuzzy-based models achieve good accuracy when applied to small- to medium-sized

requirements, their scalability and accuracy decrease significantly when the number of requirements increases. Overall, the proposed algorithm offers a robust, scalable, contextual, and semantically aware RP technique that advances the state of the art in the RE field. It lays a strong foundation for handling RD and RP processes in real-world, complex, and modern software projects by efficiently combining a deep semantic RD analysis mechanism and a strong rule-based prioritization approach.

9. FUTURE WORK

Several areas remain for further refinement:

- Integrating domain-specific taxonomies or ontologies into the proposed algorithm can increase the contextual accuracy of RD detection and prioritization.
- Development of a visual tool will enable stakeholders to adjust the requirements priority weights, conduct what-if analysis, and validate the decisions, increasing the algorithm's effectiveness.
- More expert annotations for the benchmark dataset will enhance the research community and comparative studies.
- We also aim to extend our proposed algorithm evaluation by conducting cross-domain validation across different datasets to ensure the algorithm's generalizability across different datasets.
- Moreover, we investigate the applicability in a real-world project where challenges such as incomplete requirements and dynamic prioritization arise. Addressing these gaps will bridge the gap between practical deployment and academic experimentation in industry.

10. DATA AND CODE AVAILABILITY

The dataset and the source code are available from the corresponding author for academic use only upon necessary request, as they form part of the authors' Ph.D. research.

Conflicts of interest

The authors declare that they have no conflicts of interest.

Funding

This research received no external funding.

Acknowledgement

The authors would like to thank the Faculty of Computing and Artificial Intelligence, Cairo University, for providing the research environment and support throughout this work.

References

[1] M. Daun, A. M. Grubb, V. Stenkova, and B. Tenbergen, "A systematic literature review of requirements engineering education," Requirements Engineering, vol. 28, no. 2, pp. 145–175, 2023. doi: 10.1007/s00766-022-00381-9.

[2] L. Montgomery, D. Fucci, A. Bouraffa, L. Scholz, and W. Maalej, "Empirical research on requirements quality: A systematic mapping study," Requirements Engineering, vol. 27, no. 4, pp. 183–209, 2022. doi: 10.1007/s00766-021-00367-z.

[3] A. Alazzawi and B. Rahmatullah, "A comprehensive review of software development life cycle methodologies: Pros, cons, and future directions," Iraqi Journal for Computer Science and Mathematics, vol. 4, no. 4, pp. 173–190, 2023, doi: 10.52866/ijcsm.2023.04.04.014.

[4] R. B. Svensson and R. Torkar, "Not all requirements prioritization criteria are equal at all times: A quantitative analysis," Journal of Systems and Software, 2024, doi: 10.1016/j.jss.2023.111909.

[5] A. Gupta, G. Poels, and P. Bera, "Using conceptual models in agile software development: A possible solution to requirements engineering challenges in agile projects," IEEE Access, vol. 10, pp. 131857–131873, 2022, doi: 10.1109/ACCESS.2022.3221428.

[6] F. Noviyanto, R. Razali, and M. Z. A. Nazri, "Understanding requirements dependency in requirements prioritization: A systematic literature review," International Journal of Advances in Intelligent Informatics, vol. 9, no. 2, pp. 249–272, 2023, doi: 10.26555/ijain. v9i2.1082.

[7] S. N. Rusli and R. A. Bakar, "Software requirements interdependencies: A systematic literature review on significance, techniques and challenges," International Journal of Computer and Digital Systems, vol. 16, no. 1, pp. 1–13, 2024.

[8] S. Hollerer, T. Sauter, and W. Kastner, "Risk assessments considering safety, security, and their interdependencies in our environments," in Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES), 2022, doi: 10.1145/3538969.3543814.

[9] M. Yaseen, A. Mustapha, and N. Ibrahim, "Minimizing inter-dependency issues of requirements in parallel developing software projects with AHP," Compusoft, 2019. [Online]. Available: https://ijact.in.

[10] F. Hujainah, R. B. A. Bakar, and M. A. A. Abdulgabber, "Investigation of requirements interdependencies in existing techniques of requirements prioritization," Tehnički Vjesnik, vol. 26, no. 4, pp. 1186–1190, 2019, doi: 10.17559/TV-20171129125407.

[11] W. Wang, F. Dumont, N. Niu, and G. Horton, "Detecting software security vulnerabilities via requirements dependency analysis," IEEE Transactions on Software Engineering, 2020, doi: 10.1109/TSE.2020.3030745.

[12] N. Mohamed, S. Mazen, and W. Helmy, "A comprehensive review of software requirements dependencies analysis techniques," Iraqi Journal for Computer Science and Mathematics, vol. 6, no. 3, art. 5, 2025, doi: 10.52866/2788-7421.1279.

[13] N. Mohamed, S. Mazen, and W. Helmy, "E-AHP: An enhanced analytical hierarchy process algorithm for prioritizing large software requirements numbers," International Journal of Advanced Computer Science and Applications, vol. 13, no. 7, 2022, doi: 10.14569/IJACSA.2022.0130725.

[14] R. Asyrofi, D. O. Siahaan, and Y. Priyadi, "Extraction dependency based on evolutionary requirement using natural language processing," in Proceedings of the 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), IEEE, 2020, doi: 10.1109/ISRITI51436.2020.9315489.

[15] J. Fischbach, B. Hauptmann, L. Konwitschny, and D. Spies, "Towards causality extraction from requirements," in Proceedings of the 28th International Requirements Engineering Conference (RE), IEEE, 2020, doi: 10.1109/RE48521.2020.00053.

[16] G. Deshpande, Q. Motger, C. Palomares, I. Kamra, K. Biesialska, and X. Franch, "Requirements dependency extraction by integrating active learning with ontology-based retrieval," in Proceedings of the 28th International Requirements Engineering Conference (RE), Zurich, Switzerland: IEEE, 2020, pp. 266–277, doi: 10.1109/RE48521.2020.00020.

[17] D. Mougouei and D. M. W. Powers, "A fuzzy-based optimization method for integrating value dependencies into software requirement selection," arXiv preprint, arXiv:2003.04806, 2020. [Online]. Available: https://arxiv.org/abs/2003.04806.

[18] D. Mougouei and D. M. W. Powers, "Dependency-aware software requirements selection using fuzzy graphs and integer programming," Expert Systems with Applications, vol. 167, art. 113748, 2021, doi: 10.1016/j.eswa.2020.113748.

[19] W. Guo, L. Zhang, and X. Lian, "Automatically detecting the conflicts between software requirements based on finer semantic analysis," arXiv preprint, arXiv:2103.02255, 2021. [Online]. Available: https://arxiv.org/abs/2103.02255.

[20] I. Gräßler and C. Oleff, "Automated requirement dependency analysis for complex technical systems," in Proceedings of the Design Society Conference, 2022, pp. 3099–3108, doi: 10.1017/pds.2022.189.

[21] M. C. Malik, D. Parikh, and A. Basar, "Supervised semantic similarity-based conflict detection algorithm: S3CDA," arXiv preprint, arXiv:2206.13690, 2022. doi: 10.48550/arXiv.2206.13690.

[22] J. Frattini, J. Fischbach, D. Mendez, M. Unterkalmsteiner, A. Vogelsang, and K. Wnuk, "Causality in requirements artifacts: Prevalence, detection, and impact," Requirements Engineering, vol. 28, no. 1, pp. 49–74, 2023, doi: 10.1007/s00766-022-00371-x.

[23] Y. Liu, M. Zhang, W. Zhao, and L. Chen, "Automatic requirement dependency extraction based on integrated active learning strategies," International Journal of Software Systems, vol. 8, no. 2, pp. 78–85, 2023, doi: 10.1007/s11633-023-1420-1.

[24] G. Deshpande, Requirements Dependency Extraction: Advanced Machine Learning Approaches and Their ROI Analysis, Ph.D. dissertation, University of Calgary, 2023. [Online]. Available: http://hdl.handle.net/1880/114394.

[25] T. Xu and Y. Cai, "Improved stacking ensemble learning for requirement dependency extraction," Mathematics, vol. 12, no. 9, art. 1272, 2024, doi:10.3390/math12091272.

[26] A. Gärtner and D. Göhlich, "Automated requirement contradiction detection through formal logic and large language models," Automated Software Engineering, vol. 31, no. 2, art. 45, 2024, doi:10.1007/s10515-024-00452-x.

[27] M. Alhanahnah, Y. Boshmaf, and B. Baudry, "DepesRAG: Towards managing software dependencies using large language models," arXiv preprint, arXiv:2405.20455, 2024, doi: 10.48550/arXiv.2405.20455.

[28] J. Doe and A. Smith, "RefExpo: Utilizing static analysis for extracting dependency graphs from software projects," in Proceedings of the 12th International Conference on Software Engineering, 2024, pp. 123–134. doi: 10.48550/arXiv.2407.02620.

[29] R. Sanyal and B. Ghoshal, "A hybrid approach to extract conceptual diagram from software requirements," Science of Computer Programming, vol. 239, art. 103186, 2025. doi: 10.1016/j.scico.2024.103186.

[30] T. L. Saaty, "What is the analytic hierarchy process?" in Mathematical Models for Decision Support, G. Mitra, H. J. Greenberg, F. A. Lootsma, M. J. Rijkaert, and H. J. Zimmermann, Eds. Berlin, Heidelberg: Springer, 1988, pp. 109–121. doi: 10.1007/978-3-642-83555-1_5.

[31] F. A. Bukhsh, Z. A. Bukhsh, and M. Daneva, "A systematic literature review on requirement prioritization techniques and their empirical evaluation," Computer Standards & Interfaces, vol. 69, art. 103389, 2020. doi: 10.1016/j.csi.2019.103389.

[32] M. A. Akbar, M. A. Khan, A. Imran, A. Usman, S. A. Madani, and H. Mouratidis, "Prioritization of global software requirements engineering barriers: An analytical hierarchy process," IET Software, vol. 15, no. 4, pp. 277–291, 2021. doi: 10.1049/sfw2.12022.

[33] K. Asad, A. Khan, M. Tariq, and H. Ali, "An AHP-based framework for effective requirement management in agile software development (ASD)," International Journal of Recent Innovation Trends in Computing and Communication, vol. 11, no. 10, pp. 454–463, Nov. 2023,doi: 10.17762/ijritcc.v11i10.8509.

[34] M. A. de Farias, P. R. Pinheiro, and D. P. Forte, "Prioritisation of software demands using the analytic hierarchy process method: A case study in a public organisation," in Proceedings of the Computer Science Online Conference, Cham, Switzerland: Springer, 2024. doi: 10.1007/978-3-031-70300-3_31.

[35] I. Ibriwesh, S.-B. Ho, and I. Chai, "Overcoming scalability issues in analytic hierarchy process with ReDCCahp: An empirical investigation," Arabian Journal for Science and Engineering, 2018,doi: 10.1007/s13369-018-3283-2

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[37] R. Johnston, "Standard deviation(s)," in Encyclopedia of Quality of Life and Well-Being Research, F. Maggino, Ed. Cham, Switzerland: Springer, 2023, doi:10.1007/978-3-031-17299-1_2846

[38] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength natural language processing in Python," 2020. [Online]. Available: https://spacy.io/

[39] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), vol. 1, Minneapolis, MN, USA, 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.

[40] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Transformers: State-of-the-art natural language processing," in Proc. 2020 Conf. Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP), Online, 2020, pp. 38–45. doi: 10.18653/v1/2020.emnlp-demos.6.

[41] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," arXiv preprint arXiv:1907.11692, 2019. [Online]. Available: https://arxiv.org/abs/1907.11692.

[42] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "PURE: A dataset of public requirements documents," in Proc. 25th Int. Conf. Requirements Engineering (RE), Lisbon, Portugal, Sep. 2017, pp. 502–507. doi: 10.1109/RE.2017.29.

[43] Y. L. Khaleel, M. A. Habeeb, and T. O. C. Edoh, "Limitations of deep learning vs. human intelligence: Training data, interpretability, bias, and ethics," Applied Data Science and Analysis, vol. 2025, no. 1, pp. 3–6, 2025, doi: 10.58496/ADSA/2025/002.

[44] N. Arshad, M. U. Baber, and A. Ullah, "Assessing the transformative influence of ChatGPT on research practices among scholars in Pakistan," Mesopotamian Journal of Big Data, vol. 2024, pp. 1–10, 2024, doi: 10.58496/MJBD/2024/001.

[45] H. A. Dida, D. Chakravarthy, and F. Rabbi, "ChatGPT and Big Data: Enhancing text-to-speech conversion," Mesopotamian Journal of Big Data, vol. 2023, pp. 31–35, 2023, doi: 10.58496/MJBD/2023/005.