The Journal of Systems and Software 211 (2024) 111981

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Extracting goal models from natural language requirement specifications ☆

Souvick Das [c,*], Novarun Deb [b], Agostino Cortesi [c], Nabendu Chaki [a]

[a] University of Calcutta, JD 2 Sector 3, Kolkata, 700106, West Bengal, India
[b] Indian Institute of Information Technology, Vadodara, Block 9, Sector 28, Gandhinagar, 382028, Gujarat, India
[c] Ca' Foscari University, 155, Via Torino, Mestre, 30170, Venezia, Italy

ARTICLE INFO

Dataset link: https://github.com/svk-cu-nlp/NLR-to-Goal-Model

Keywords:
Natural language requirements
Natural language processing
Transformer model
Entity type recognition
Contextual vector
Synonymy vector

ABSTRACT

Unstructured (or, semi-structured) natural language is mostly used to capture the requirement specifications both for legacy software systems and for modern day software systems. The adoption of a formal approach to the specification of the requirements, using goal models, enables rigorous and formal inspections while analyzing the requirements for satisfiability, consistency, completeness, conflicts and ambiguities. However, such a formal approach is often considered burdening for the analysts' activity as it requires additional skills, and is therefore, discarded a priori. This works aims to bridge the gap between natural language requirement specifications and efficient goal model analysis techniques. We propose a framework that uses extensive natural language processing techniques to transform a set of unstructured natural language requirement specifications to the corresponding goal model. We combine techniques such as parts-of-speech tagging, dependency parsing, contextual and synonymy vector generation with the FiBER transformer model. An extensive unbiased crowd-sourced evaluation of the proposed framework has been performed, showing an acceptability rate (total and partial combined) of 95%. Time and space analyses of our framework also demonstrate the scalability of the proposed solution.

1. Introduction

The evolution of software development methodologies – from plan-based Waterfall models (in the 70s and 80s) to Agile and Lean Development models (in the late 90s and early 2000s) and, the more recent, DevOps techniques (popularized over the last decade) – has constantly highlighted the need for the software industry to respond to volatile business environments more rapidly and effectively. This has resulted in a paradigm shift from investing in extensive documentations to rapid delivery of working code. As a result, both maintenance of legacy systems and development of modern software suffer from one common flaw — the negligence of requirements engineers in applying proper requirements engineering techniques to manage the volatile requirements of the business and software application effectively. In this regard, requirements elicitation and analysis is one of the most critical activities for ensuring the correctness and efficacy of the software being developed and maintained.

Requirement goal models have the advantages of goal refinement formalization (Darimont and Van Lamsweerde, 1996) as well as reusability (Mussbacher et al., 2007; Duran et al., 2015). The goal-oriented requirements engineering community has proposed frameworks, tools and solutions for effective management, analysis and validation of system requirements with the help of requirement goal models (Van Lamsweerde, 2009, 2001a). Most requirement specifications are captured – both within legacy requirement specification documents and agile story cards – using natural language statements. Natural language specifications are prone to inconsistencies, incompleteness, and errors arising out of incorrect interpretations (Popescu et al., 2007; Landhäußer et al., 2014; Kof, 2005). Automatic techniques cannot be applied on such specifications for performing different types of analyses such as satisfiability analysis, entailment checking, inconsistency checking, and others.

Existing works in the literature have already demonstrated how requirements models can be derived from *structured* or *semi-structured* natural language requirement specification documents. Most of these works aim to derive UML Use Case or Sequence diagrams from semi-structured natural language requirements (Deeptimahanti and Sanyal, 2011; Kumar and Sanyal, 2008; Deeptimahanti and Babar, 2009a; Güneş and Aydemir, 2020). On the other hand, deriving requirement goal models from *unstructured* natural language requirement (uNLR) specifications is still an open research problem.

In this paper, our objective is to create and empirically assess a comprehensive framework for managing system requirements, thereby

☆ Editor: Liu Xiao.

* Corresponding author.
E-mail addresses: souvick.das@unive.it (S. Das), novarun_deb@iiitvadodara.ac.in (N. Deb), cortesi@unive.it (A. Cortesi), nabendu@ieee.org (N. Chaki).

https://doi.org/10.1016/j.jss.2024.111981
Received 3 November 2022; Received in revised form 31 August 2023; Accepted 19 January 2024
Available online 30 January 2024

filling the noted research gap. The end objective of this exercise is to extract the goal modeling constructs that remain embedded within natural language requirement specifications. The derivation of goal models from detailed natural language requirement specifications can be labor intensive and prone to errors that arise out of misinterpretation. The research is quite challenging as it involves extensive text analysis tasks. In this paper, we try to address these challenges by combining natural language processing (NLP) techniques and methods like Natural Language Understanding (NLU), Information Retrieval (IR), Sentence Embedding, Dependency Parsing, Entity Type Recognition, and Similarity Checking, in addition to several pre-processing tasks as well. Recent progresses in the NLP domain allow us to reduce human intervention and incorporate as much automation as possible.

The proposed comprehensive end-to-end framework is composed of several modules and algorithms that utilize different NLP techniques to facilitate the process of deriving a goal model from uNLR specifications. At first we identify goal modeling constructs like goals, softgoals, and resources, along with the semantic relationships among them as given in the uNLR document. In the next phase, the framework employs two algorithms that utilize the information from the previous stages and generate the goal model. The goal model construction process consists of three sub-processes - *(a)* goal model construction with actor boundaries; *(b)* representing goal decompositions; and *(c)* representing hard goal and softgoal associations. The framework uses machine learning techniques to address specific challenges during the goal model derivation process.

We evaluated the application of our framework on two different case studies: (1) *a meeting scheduling system* and (2) *an online shopping system*. We achieve over 85% of accuracy in all the tasks - *Goal Identification, Goal Decomposition, Resource Identification, Goal-Resource Association, Softgoal Recognition* and *Goal-Softgoal Association* - required to construct the goal models for the above use cases. We adopted a crowd-sourcing based evaluation approach for estimating the acceptability of our solutions tailored for the requirements engineering community. Time and space analyses also demonstrate the scalability of the proposed solution.

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

1. An end-to-end framework for generating goal models from unstructured natural language requirements, supporting goal decompositions, and associations among goals, resources and softgoals.
2. A software requirements classification approach to classify functional and non-functional requirements(NFR) with 92% of accuracy that outperforms state-of-the-art works for requirements classification. The classification model is also able to classify the types of NFR.
3. A working prototype[1] of our proposed framework that allows the research community and requirements engineers to use our framework in real world software-enabled business environments.

The rest of the paper is organized as follows. Section 3 presents the existing state-of-the-art for deriving goal models from natural language requirements. Section 2.5 gives a brief idea of how goal model entities and relationships can be derived from uNLR specifications. In Section 2.4, we document the preliminary concepts of goal models and different NLP mechanisms. The framework is presented in full details in Section 4. In Section 5, we experimentally evaluate it on two well-known case studies. Section 7 concludes.

[1] https://github.com/svk-cu-nlp/NLR-to-Goal-Model.

2. Background, related concepts and motivating example

This work makes a conscious effort to bring together two very diverse research communities — the goal-oriented requirements engineering community and the natural language processing community. In order to make the paper better readable for this diverse target audience, we would like to briefly discuss the concepts which have been adopted from these two communities for the presentation of our research work. We also present a simple example to demonstrate how the concepts may be used to achieve the desired output.

2.1. Representations of natural language requirements

The requirements engineering community has proposed different notations for documenting system requirement specifications such as structured, semi-structured and unstructured natural language, design description languages, graphical notations, and mathematical specifications. However, among these the two most popular and widely used notations are as follows:

- *Structured Natural Language Requirements:* The requirement specifications are expressed in plain English on a common form or template. Each field contains information on a different aspect of the requirement. Several templates (Rupp et al., 2009; Arora et al., 2015) have been proposed in the requirements engineering literature to be served as a simple tool for improving the quality of requirements by avoiding complex structures, ambiguities, and inconsistencies in requirements. Such notations have been widely popularized by Agile teams who use them for capturing user stories or story cards (Lucassen et al., 2016; Dalpiaz et al., 2019; Spijkman et al., 2021).
- *Unstructured Natural Language Requirements:* The more popular notation for capturing requirement specifications in the industry is the unstructured natural language notation. Such specifications, on the other hand, might be complicated and difficult to interpret since the requirements are subject to the interpretation of the user. There are three main problems that often arise when requirements are written in unstructured natural language sentences (Jackson, 1995):
  - *Lack of clarity:* It is sometimes difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read.
  - *Requirements confusion:* Functional requirements, non-functional requirements, system goals and design information may not be clearly distinguished.
  - *Requirements amalgamation:* Several different requirements may be expressed together as a single requirement.

Within the scope of this paper, our emphasis lies in employing NLP techniques to process unstructured natural language requirement (uNLR) specifications and subsequently generating modeling constructs, particularly tGRL (Abdelzad et al., 2015b) goal models, from these specifications.

2.2. Concepts of goal model

The goal oriented requirements engineering literature has introduced many different goal modeling concepts over the past couple of decades. In this section, we highlight and briefly explain only those concepts which we have used in this paper and which are fundamental to most goal modeling frameworks that are available in the existing state-of-the-art.

- *Actor:* An actor is an active entity that carries out actions to achieve goals by exercising its knowledge (Yu, 2011).

- *Goal:* A goal is a condition or state of affairs in the world that the actor would like to achieve (Yu, 2011). They are binary in nature i.e. they can be satisfied or not satisfied. Example - "User must enter shipping address before placing order"
- *Softgoal:* A softgoal is a condition in the world which the actor would like to achieve but unlike in the concept of a hard goal, the criteria for the condition being achieved is not sharply designed a priori, and is subject to interpretation (Yu, 2011). They are fuzzy in nature. Example — Reliability, Performance, Security.
- *Goal Decomposition:* Goals are refined into subgoals that elaborate on how the goal is achieved. Goal decomposition could be of two types.
  - *AND Decomposition:* AND-refinement links relate a goal to a set of subgoals (called refinement); this means that satisfying all subgoals in the refinement is sufficient for satisfying the parent goal (Van Lamsweerde, 2001b). For example — The goal "Place order" can be decomposed into "Provide delivery address" and "Make payment".
  - *OR Decomposition:* OR-refinement links relate a goal to an alternative set of refinements; this means that satisfying one of the refinements is sufficient for satisfying the parent goal (Van Lamsweerde, 2001b). For example — The goal "Login" can be achieved by using "Facebook authentication" or "Google authentication".
- *Resource:* Physical or informational entity (Yu, 2011).
- *Means-end Link:* A means-ends link indicates a relationship between an end – which can be a goal to be achieved, a task to be accomplished, a resource to be produced, or a softgoal to be satisficed – and a means for attaining it (Yu, 2011). For example — The goal "Encrypting data" has a means-end link towards satisficing the softgoal "Data Security".

2.3. Challenges of constructing goal models from uNLR specifications:

The construction of goal models from unstructured natural language requirements brings different critical challenges. This section highlights the major challenges we encountered while developing our framework.

- *Understanding the Structure of Sentences:* Extraction of different components like actors, goals, and resources from uNLR statements requires a deep understanding of the English grammatical structure. Identifying subject, verb, object and relations among multiple words within an active (or passive) sentence involves a moderate level of NLP tasks like POS Tagger and Dependency Parser.
- *Identifying Actors and Resources:* Actors and resources are entities which are always domain-specific in nature. Identifying actors and resources from the noun phrases of a sentence brings another significant challenge.
- *Identifying Goals and Softgoals:* In goal models, functional requirements are represented as goals (also called hard goals) and non-functional requirements are represented as softgoals. Several key concepts of a goal model like goal decomposition, the contributions of softgoals, and dependency among goals and softgoals define the completeness of a goal model. This is the reason why the recognition of goals and softgoals is so crucial. In order to identify the goals and softgoals in the goal model, we must classify the requirements. Requirements classification task can classify the requirements into two categories (i) Functional requirements and (ii) Non-functional requirements. We can further extend the classification task to classify sub categories of non-functional requirements.
- *Goal Decomposition:* In natural language requirements, the decomposition of goal may not be specified in an explicit manner. The decompositions of goals need to be extracted from multiple uNLR statements. Semantic relationships among requirement statements with respect to the same goal is the key to identify the goal decomposition.
- *Decomposition Type Identification:* Once we identify the existence of a potential goal decomposition across a set of uNLR statements, we need to also identify the type of decomposition. A supervised classification task to classify the decomposition type could be the possible solution.
- *Softgoal-Goal Association:* Goals can contribute some positive or negative values towards satisficing (also referred to as satisfying) a softgoal. In this work, we aim to identify only the association between goals and softgoals; predicting the contribution value is beyond the scope of this work. Identifying the association between softgoals and goals may require a systematic approach. The systematic approach could leverage NLP tasks like uNLR classification, and semantic similarity between multiple requirements to accomplish the identification of the goal and softgoal association.

In this paper, we take up all the above mentioned challenges and try to solve them by combining different NLP techniques.

2.4. Natural language processing tasks

This section briefly elaborates the NLP concepts and technologies that have been used in our proposed framework. This section is aimed for the requirements engineering community readers who might not be familiar with NLP concepts.

- *Dependency Parsing.* A dependency parser analyzes the grammatical structure of a sentence, establishing relationships among the words of the sentence. It maintains a partial parse – a stack of words that are presently being processed and a buffer of words that are yet to be parsed – at every step. An open-source python library spaCy[2] has been used in this work to generate a dependency parse tree. The spaCy dependency parser offers tokens with different properties to traverse the dependency parse tree. In this work, we have used the Stanford dependency parser models in spaCy pipeline. It is worth mentioning that the Stanford dependency parser models (Chen and Manning, 2014; Dozat et al., 2017) support English (with Universal Dependencies, Stanford Dependencies and CoNLL Dependencies) and achieve the highest accuracy in the CoNLL 2017 (Straka and Straková, 2017) and 2018 (Zeman et al., 2018) shared task.
  Natural language requirement specifications can be diverse in structure, as they can be written in either active or passive voice. Our framework uses a dependency parser to efficiently recognize the structure of these sentences, and to find the subjects, verbs, and objects within them. This allows us to accurately parse natural language requirements specifications, regardless of their structure.
- *Information Extraction* (Cowie and Lehnert, 1996). Information extraction is a powerful natural language processing (NLP) technique that can be used to extract structured information from unstructured text. This capability proves invaluable in extracting crucial components from requirements specifications for constructing comprehensive goal models. The community might be using the information extraction techniques to extract entities and establishing links between different pieces of information. In our framework, information extraction plays a vital role in the implementation of several components. Resource identification, for example, requires more deep analysis of the text. Entity type recognition is the key to identify resources specified in the software requirements specification document. These extracted information are the building blocks for the rest of the framework.

[2] https://spacy.io/usage/linguistic-features#morphology.

Table 1
Specification of parameters to fine-tune the FiBER model for different tasks.

Task	Parameters	Value
1. Actor and resource recognition	Optimizer	AdamW
	Learning rate	1e−3
	Loss function	Cross-entropy
	Weight decay	0.01
	Epoch	8
	Batch size	16
	Train and test split	0.80
2. Goal decomposition type prediction	Optimizer	AdamW
	Learning rate	5e−05
	Loss function	BinaryCrossEntropy
	Weight decay	0.01
	Epoch	10
	Batch size	16
	Train and test split	0.8
3. Goal softgoal classification	Optimizer	AdamW
	Learning rate	5e−05
	Loss function	Cross-entropy
	Weight decay	0.01
	Epoch	10
	Batch size	16
	Train and test split	0.75

- *Sentence Embedding Model*. The quality of understanding a sentence is determined based on three characteristics of a word embedding or sentence embedding model. These are as follows:
  - The corpus on which the model is trained.
  - The architecture of the model.
  - The amount of contextual knowledge focused on by the model.

  The *FiBER* (Das et al., 2021) model is a transformer architecture based language model which is fine-tuned on the BERT (Devlin et al., 2018) model with the PURE (Ferrari et al., 2017) dataset (comprising of unstructured natural language requirement statements). The *FiBER* model uses the vast vocabulary of BERT, grasps certain words from the domain of requirements engineering and has the ability to generate sentence embeddings. The *FiBER* model comes with very high accuracy of 88% for identifying both similar and dissimilar natural language requirements. In different phases of our framework, identifying related requirement statements is very crucial and cosine similarity make it easier to achieve relatedness measures. NFR classification task is another vital feature in our framework. We have also trained the *FiBER* model for NFR classification task and it achieves 89% of accuracy for multi-class classification task. The *FiBER* model has also been used for the *Decomposition Type* prediction task of the framework. Moreover, the FiBER model has been trained for identifying goal model entities, more specifically *Actors* and *Resources*. Table 1 shows different parameters used to fine-tune the FiBER model for different tasks.

2.5. A motivating example

We would like to introduce a preliminary case study that allows the reader to better appreciate the exact problem statement and the desired outcomes from our proposed solution. The requirement statements are kept simple, intentionally, in order to demonstrate the entire process with ease. It is worth mentioning here that this example is not part of the evaluation of our framework. It only explains the different outputs that the system is expected to generate at every step and may differ with actual evaluation outputs. We consider a partial set of requirements for an e-commerce application taken from the PROMISE (Cheikhi and Abran, 2013) dataset. Our framework performs the following set of activities.

Table 2
uNLR specifications (partial) for E-commerce application.

Functional requirements	1. User shall login into the system. 2. User can use either Google credentials or Facebook credentials in order to login. 3. User shall search products. 4. User has to enter text to search product. 5. User can filter searched products by category, price or color. 6. User shall place order of products. 7. User can make payment through Net Banking or credit card. 8. User must provide delivery details like street address, zip code and mobile number.
Non-functional requirements	9. User login credentials must be stored using encryption. 10. Payment should be done on secure channel.
Constraints	11. User must make payment before placing order. 12. User has to login before ordering products.

1. *Classification of Requirements Statements.* The entire process starts with analyzing the requirements statements and identifying functional requirements, non-functional requirements and constraints as depicted in Table 2. Functional and non-functional requirements must be classified in order to identify goals and softgoals for goal model construction.
2. *Goal Model Component Extraction*. At this point, we need to analyze every uNLR statement in order to extract different building blocks of a goal model such as Goals, Actors, Resources and Softgoals. We aim to identify goal decompositions from single requirement statements or multiple consecutive requirement statements. Furthermore, we intend to infer relationships among multiple functional and non-functional requirements in terms of goal-softgoal associations. Table 3 gives an overview of how essential components may be extracted from the partial set of uNLR statements listed in Table 2. Additional information, like dependencies among goals and softgoals can also be observed for the 9th and 10th requirements in Table 3.
3. *Goal Model Generation.* Once we have all the necessary information for generating the goal model, we aim to generate the goal model description using the tGRL (Abdelzad et al., 2015b) domain specific language (DSL). Goal model generation from the components is significantly challenging. We propose a couple of algorithms to arrange the extracted components within a goal model and the corresponding tGRL description should be generated as shown in Fig. 1. We choose tGRL over other goal model approaches because the DSL is simple to understand and automated composition of the goal model is relatively easy. The tGRL description can also be visualized graphically using the existing jUCMNav tool (Abdelzad et al., 2015a).

3. Related works

The strong relationship between Natural Language (NL) and software requirements has inspired researchers to introduce NLP to automate the processing of natural language requirements (NLR) for later stages of software development. Basic tools and libraries like POS(Parts-of-Speech) tagger (Santorini, 1990), dependency parser, lemmatizer, entity recognizer are often used in analyzing natural language requirements to build requirements models (Sagar and Abirami, 2014; Robeer et al., 2016; Letsholo et al., 2013). Word and sentence embedding models like Word2Vec (Mikolov et al., 2013), BERT (Devlin et al., 2018) and other language models (Liu et al., 2019; Sanh et al., 2019) are also gaining attention as they interpret natural languages into machine readable form and help to perform critical tasks like Requirements Dependency Classification(RDC) (Deshpande et al., 2021), requirements

Table 3
Extracted components of natural language requirements.

Natural language requirements	Type of requirements	Extracted entities and relationships
1. User shall login into the system.	Functional requirements	Actor: User, Goal: Login
2. User can use either Google credentials or Facebook credentials in order to login.	Functional Requirements	Actor: User, Goal: Login, Subgoal: Login via Google Credentials, Login via Facebook Credentials, Decomposition Type: OR
3. User shall search products.	Functional requirements	Actor: User, Goal: Search Products
4. User has to enter text to search product.	Functional requirements	Actor: User, Goal: Search Products,Subgoal: Enter text, Decomposition Type: AND
5. User can filter searched products by category, price or color.	Functional Requirements	Actor: User, Goal: Filter Searched Products, Subgoal: Filter by category, Filter by price, Filter by color, Decomposition Type: OR
6. User shall place order of products.	Functional requirements	Actor: User, Goal: Place product order
7. User can make payment through Net Banking or credit card.	Functional requirements	Actor: User, Goal: Make payment, Subgoal: Payment through Net Banking, Payment through Credit card, Decomposition Type: OR
8. User must provide delivery details like street address, zip code and mobile number.	Functional requirements	Actor: User, Goal: Provide delivery address details, Subgoal: Provide street address, Provide zip code, Provide mobile number, Decomposition Type: AND
9. User login credentials must be stored using encryption.	Non-functional requirements	Actor: User, Softgoal: encryption, Associated Goal: Login
10. Payment should be done on secure channel.	Non-functional requirements	Actor: User, Softgoal: Security, Associated Goal: Payment.
11. User must make payment before placing order.	Constraint	Actor: User, Dependent Goal: Place order, Dependee Goal: Make payment.
12. User has to login before ordering products.	Constraint	Actor: User, Dependent Goal: Order products, Dependee Goal: Login

```
tgrl {
    actor User {
        goal Login {
            subgoal: 'Login via Google Credentials', 'Login via Facebook Credentials'
            decompositionType: 'OR'
            softgoal: 'Encryption'
        }
        goal Search Products {
            subgoal: 'Enter text'
            decompositionType: 'AND'
        }

        goal Filter Searched Products {
            subgoal: 'Filter by category', 'Filter by price', 'Filter by color'
            decompositionType: 'OR'
        }
        goal Place order {
            dependedOn: 'Make Payment', 'Login'
        }
        goal Make payment {
            subgoal: 'Payment through Net Banking', 'Payment through Credit card'
            decompositionType: 'OR'
            softgoal: 'Security'

        }
        goal Provide delivery details {
            subgoal: 'Provide street address', 'Provide zip code', 'Provide mobile number'
            decompositionType: 'AND'
        }
        softgoal Security;
        softgoal Encryption;
    }
}
```

Fig. 1. tGRL description of the goal model.

classification (Hey et al., 2020), entity coreference identification (Agarwal et al., 2019) and many more. Automated model generation from NLR text documents is an active field of study for researchers. Several researches like Kumar and Sanyal (2008), Zhou and Zhou (2004) and Ben Abdessalem Karaa et al. (2016) have been conducted to generate UML class diagram from NLR specification documents. Similar researches (Deeptimahanti and Babar, 2009a; More and Phalnikar, 2012; Ibrahim and Ahmad, 2010) have been introduced to develop UML use case diagram from NLR documents using NLP tools and techniques. Kochbati et al. (2021) proposes a machine learning-based approach to automatically break down a software system into sub-systems and generate preliminary architecture models from natural language user stories. The paper uses word2vec for semantic similarity and optimal cluster estimation. It proposes a clustering solution to generate UML

Table 4
Comparison of existing systems.

Paper	Human intervention needed	Classification of requirements considered	Softgoal identified	Unstructured NLR considered	Final outcome	Acceptance analysis	Evaluation metric
Letsholo et al. (2013)	✓	✗	✗	✓	Analysis model on UML class diagram	✗	Precision and recall
Robeer et al. (2016)	✗	✗	✗	✗	Conceptual models	✗	Precision and recall
Deeptimahanti and Babar (2009b)	✓	✗	✗	✗	UML class diagram, analysis model and collaboration diagram	✗	Not evaluated
Sagar and Abirami (2014)	✗	✗	✗	✓	UML class diagram	✗	Precision and recall
Ben Abdessalem Karaa et al. (2016)	✗	✗	✗	✓	UML class diagram	✗	Precision and recall
Cleland-Huang et al. (2006)	✓	✓	✓	✓	Visualize the quality concerns	✗	Precision recall and F1-measure
Güneş and Aydemir (2020)	✗	✗	✗	✗	Goal models	✗	Not evaluated
Arora et al. (2016)	✓	✗	✗	✓	Domain models	✓	Acceptance analysis
Shimada et al. (2017)	✓	✗	✗	✓	Goal Models	✓	Acceptance analysis
Bragilovski et al. (2022)	✓	✗	✗	✓	UML class diagram	✓	Acceptance analysis
Zhou et al. (2022)	✓	✗	✗	✓	Goal model snippets	✗	Precision recall and F1-measure
Proposed approach	✓	✓	✓	✓	Goal models	✓	Acceptance analysis

use-case models from user stories. In another work (Sagar and Abirami, 2014), authors presented an automated system that involves different NLP tools and several design rules to create a class diagram from natural language requirements. They have also classified relationships as Associations, Aggregation, Composition and Generalization. Letsholo et al. (2013) developed the TRAM tool platform to construct analysis models automatically from the natural language specifications by incorporating conceptual patterns with NLP techniques. In another research, Arora et al. (2016) proposed an automated approach of building NL Domain Models based on the use of existing NLP tools, and are also proposing new modeling rules using NLP Dependence Parser. Robeer et al. (2016) introduce a completely automatic tool to create a conceptual model as OWL ontology from user stories by using NLP heuristics. Another work (Bragilovski et al., 2022) offers example-based guidelines for deriving class and use case diagrams from user stories. Through a controlled experiment with 77 undergraduate students, it demonstrates that these guidelines improve the completeness and validity of conceptual models for medium complexity cases, potentially assisting analysts in refining user stories.

Weber-Jahnke and Onabajo (2009) proposed an approach for the mining of safety goals and the analysis of the goal model. Additionally, they have provided the tool support for the framework. Cleland-Huang et al. (2006) proposed a machine learning and data mining based approach of eliciting quality concerns and constructing a Softgoal Interdependency Graph (SIG) from a natural language requirements specification. In another work (Casagrande et al., 2014), authors have integrated data mining and NLP approaches with KAOS framework to extract goals from textual data. In semi-automatic and iterative format the NLP-KAOS system is able to generate a goal specification. Another research (Liaskos et al., 2010) proposed an approach which uses NLP to identify stakeholder's preferences from natural language expressions given by stakeholders themselves. Later on, they have associated these preferences with the corresponding goal of the goal model. A framework for goal model construction from user stories using NLP techniques and some heuristics has been presented by Güneş and Aydemir (2020). Shimada et al. (2017) presented an approach to construct goal model from requirements descriptions. The approach consists of three steps — requirements decomposition, goal extraction and integration. First, the requirements description is decomposed into fragments. Several extraction rules have been applied to elicit goals from the natural language requirements. Finally, different small goal models are combined into one goal model. Zhou et al. (2022) present a systematic framework that semi-automatically generates goal model snippets from textual requirements specifications. They fine-tune the BERT model on datasets for actor entity extraction, intention entity extraction, and actor relation extraction. This interactive and iterative modeling process effectively combines human decisions and AI algorithms, offering valuable assistance in performing iStar modeling. In another research (Bhatia et al., 2016) the authors have proposed a hybrid combination of crowd sourcing and NLP to extract privacy goals from the privacy policies. In this framework, crowd workers interprets phrase-level policy as small tasks. In order to keep the tasks small and affordable, dependency parsing based on part-of speech (POS) tagging has been incorporated to annotate privacy goals semi-automatically. In Table 4, the state-of-the-art approaches in conceptual domain model generation from natural language requirements are highlighted.

The current state-of-the-art shows several approaches or frameworks leveraging NLP techniques and rule based approaches to obtain goal model from NLR. However, these methods work on structured or semi-structured NLR and covers few concepts of goal model to incorporate. This domain still lacks a framework enriched with robust NLP techniques to understand unstructured NLR to work on and covers some crucial concepts of goal model while developing the model. In comparison with previous works presented in this area, our framework takes a step further. Our framework involves multiple NLP techniques to address those critical aspects of constructing goal models from unstructured natural language requirements.

4. Goal model generation

In this section, we introduce our framework, which takes unstructured natural language requirements (uNLRs) as input and uses several natural language processing (NLP) tasks on it. As an end product of the framework, we re-engineer the goal model corresponding to the uNLRs provided as input.

Fig. 2 depicts the overall architecture, consisting of different modules, marked using numbers for sequencing purposes, and data stores. Multiple machine learning algorithms have been used to accomplish challenging tasks mentioned in Section 2.3. In the following subsections, we elaborate on each of these modules and explain how different NLP approaches can be adopted and combined to accomplish the desired tasks and overcome different research challenges. We consider the

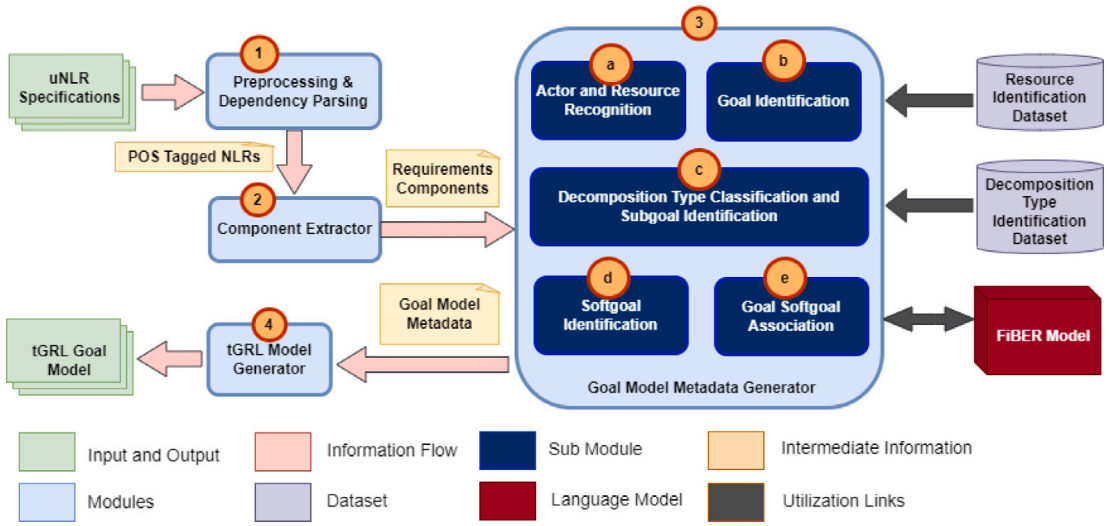


Fig. 2. Framework of goal model generation from NLRs.

following three sample uNLR statements (from the PROMISE[3] dataset) to illustrate how every module works and the intermediate outputs that are generated towards producing the final goal model.

1. The system uses a standard navigation menu familiar to most web users.
2. The system shall allow user to navigate to different menus of the WebApp.
3. The system must be able to export reports in spreadsheet form as xls or 123 formats.

4.1. Preprocessing and dependency parsing

Applying different machine learning methods on raw uNLR statements is quite difficult and the performance is unsatisfactory. For better outcomes, the very first task that needs to be performed on such statements is preprocessing. The first module (marked as 1 in Fig. 2) represents this module. The preprocessing task which is carried out for almost every NLP approach is parts-of-speech (POS) tagging. POS tagging is useful to understand the construction of the language. Fig. 3 shows the POS tagging of our sample uNLR sentences where different labels represent different part-of-speeches. For example *NN* represents Noun, singular or mass, *VB* represents verb in base form, whereas *VBZ* represents verb in 3rd person singular present form, *JJ* represents Adjective and so on. Detailed list of these parts-of-speech tags can be found in Penn treebank[4] and NLTK guide.[5] Finding dependencies among different words within a sentence becomes easier when POS tagging is used. In our framework, POS tagging is useful for identifying different components of uNLR statements. We use the Stanford POS-tagger (Santorini, 1990) which is able to achieve 96.97% of accuracy in identifying parts-of-speech accurately.

On the other hand, the Stanford dependency parser (Chen and Manning, 2014) is used to obtain dependency among words in terms of subject, verb and predicate. For example — in the first sentence, the word *system* has the subjective (nsubj) relation with the verb *allow* and the object is user. The main verb *allow* is connected using *clausal complement* (xcomp) relationship with another verb *navigate*. The detailed explanation of different relationships is present in stanford's documentation.[6] Fig. 4 shows the output as a dependency tree generated using the dependency parser.

4.2. Component extraction

The POS tagged statements along with dependency relationship information are accepted by the *Component Extractor* module which is the second component of the framework (marked as 2 in Fig. 2). This module identifies subject, the object (or the predicate) and the action or process (the verb) by establishing links among the words based on their type of relations. It is worth mentioning that the dependency parser takes care of passive sentences also and extracts the subject, object and verb. Fig. 4 shows that we can precisely extract the noun words or nominal compounds from the subjects and objects of the sentences, respectively. For example — in the second sentence *The system* has the subjective relation with the main verb *uses*. The main verb is again connected with nominal compound *navigation menu* using *obj* relation. Furthermore, the nominal compound has the adjective attached to it using *amod* (adjectival modifier) relation. These extracted information is essential for the next most crucial module of the framework. The next module is responsible of identifying the goal model artifacts – *actors*, *goals*, *resources*, and *softgoals* – and the relationships between them.

4.3. Goal model metadata generation

At this point, we have all the components extracted from our uNLR specifications that are required to generate the goal model metadata corresponding to the specifications. One major aspect that still remains to be explored is the actual binding of these components to create the goal model. The module of the framework (marked as 3 in Fig. 2) represents the goal model metadata generation module. This module tries to identify the goal model artifacts and their relationships across multiple uNLR statements. We define the relationships by identifying goal decompositions, associations between goals and softgoals and those between goals and actors.

[3] http://promise.site.uottawa.ca/SERepository/.

[4] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

[5] https://www.nltk.org/book/ch05.html.

[6] https://web.stanford.edu/jurafsky/slp3/14.pdf.

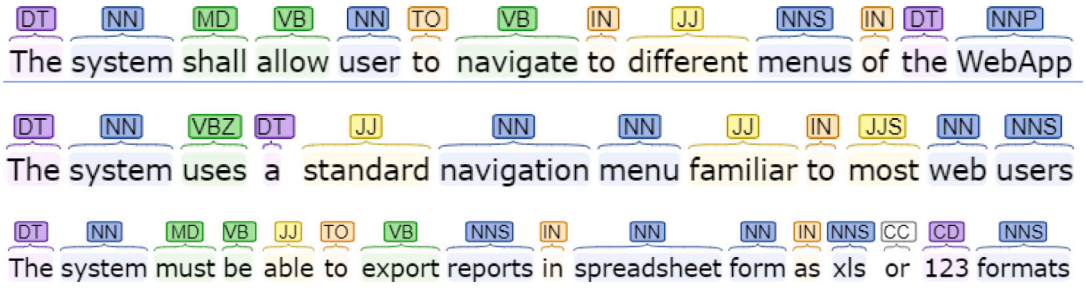


Fig. 3. Example uNLRs tagged with POS.

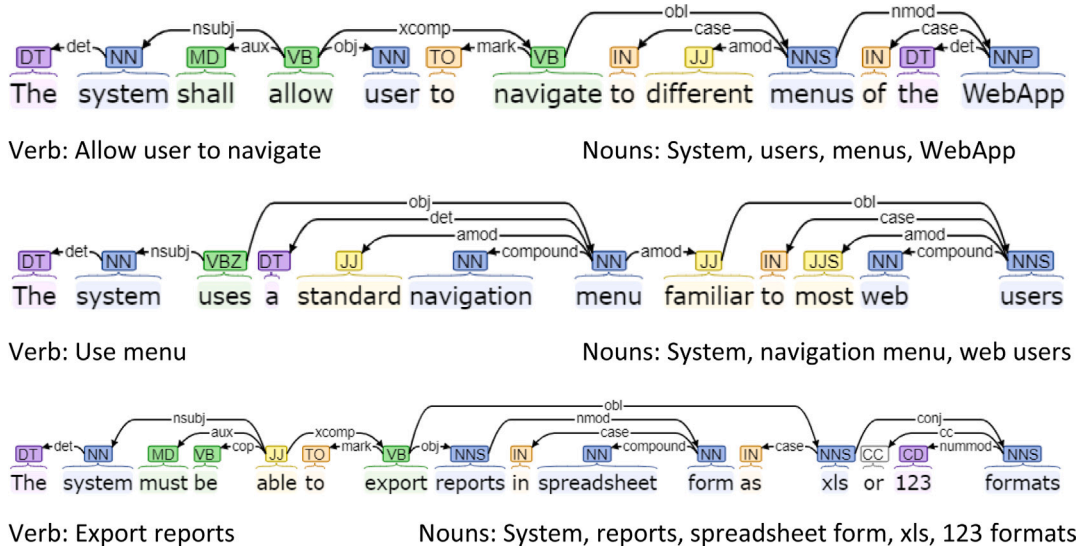


Fig. 4. Dependency parsing and goal identification for three example uNLRs.

4.3.1. Actor and resource recognition

One of the major challenges involved in the metadata generator module is to identify *actors* and *resources* from the uNLR specifications. The metadata generator module (marked as 3a in Fig. 2) uses entity type recognition for this purpose. The Component Extractor module has already extracted the nouns and nominal compounds from the natural language sentences. We need to determine the *actor* and *resource* from the noun (or nominal compound) entity. For example, we may extract a nominal compound like "*Mainframe System*", "*System Administrator*". It is not easy to recognize a noun phrase as a resource because every project has its own set of domain specific resources. It is hard to build a universal model with the ability to recognize resources from different projects. We have prepared a dataset containing requirement statements along with labeled resources. We build a classification model and train it on the above dataset. The details of the dataset, the classification model, and how it is used for the recognition tasks are elaborated as follows:

(a) ***Dataset Generation:*** We have taken uNLR statements from the PROMISE_exp (Lima et al., 2019) dataset. PROMISE_exp is the expansion of the original PROMISE (Cheikhi and Abran, 2013) dataset and contains 969 requirements statements from 34 different projects. We further expand this uNLR dataset by adding 3000 sentences containing resources related to IT infrastructure domain. These sentences are sourced from diverse data, including the PURE dataset,[7] user guides of various software applications, articles on operating systems (e.g., official documentation of Windows, Mac), databases (e.g., official documentation

Table 5
Software requirement specific entities used for the annotation of the dataset.

Entity	Description	Example
O	Non Entity Tokens	Tokens not belong to entity set
VART	Virtual Artifacts	Case History, Record, Account
PART	Physical Artifacts	Credit Card, Course Material, Brochure
SSW	Standard Software Components	Anti-virus, URL, PDF, JPG, Database
GUI	Components of Graphical User Interface	Widget, Page, Applet, Wizard
HW	Hardware Components	Printer, Display, Projection Screen, Pump Regulator
PLAT	Application or Software Platforms	Browser, CRM, ERP, Ubuntu
EVE	Event or Activity	Screening, Beta-Testing,
TIM	Time Indicator	Minute, Second, Hour, Date, Day, Month, Year
ORG	Organization	Apple, Microsoft, IBM
ACTOR	Entity which has a role	Admin, Developer, Client, System

[7] https://zenodo.org/record/1414117#.ZAr4u3bMK3A.

of MongoDB, Oracle), cyber security (MITRE documentation), and software product descriptions, including AWS documentation. textitAnnotation Process: Throughout the dataset creation process, a team of five annotators, including two authors of the paper, actively participated in the annotation task. Each annotator was responsible for annotating approximately 800 sentences, which were subsequently distributed among the other annotators for further checks. In order to ensure the quality and accuracy of the dataset, validation was performed collectively by all annotators and authors. The majority voting consensus was used to finalize the labels for each entity within the sentences. The process of annotation is accomplished with the help of *ner-annotator* tool.[8]

In Table 5, we present a diverse set with ten (10) entity labels – nine (9) entity categories and one (1) non-entity label – which covers different aspects of the software specifications. The identified nine entity categories can help end users to extract information from the uNLR specification documents. We manually identify the resources and label them after initial data cleaning and pre-processing steps. No lemmatization or stemming technique has been applied as we opted to keep the semantic structure of words similar to the documents prepared in the software development processes. In order to create the annotations, we apply the widely accepted BIO representation scheme.[9] Table 6 shows an example of a uNLR sentence from the dataset which has been annotated in the BIO format. Finally, we examine the annotations manually in order to ensure high-quality data.

(b) ***Entity Recognition Model:*** Next, we define the model for entity classification task. We load the FiBER model (Das et al., 2021) and additionally specify the number of labels. The base layers are initialized with pre-trained weights. The token classification head on top has randomly initialized weights. We use these random weights and pre-trained weights together to train the model using our labeled dataset. We use cross entropy loss function (Chong and Zak, 2013) and AdamW optimizer (Loshchilov and Hutter, 2017) with default learning rate of 1e−3. (Please refer to Task 1 in Table 1). We evaluate the performance of the classifier on 20% of test data and it shows 92% of accuracy, 92% of precision, 91% of recall and 92% of F1-Score. Finally, we save the model by saving the vocabulary, model weights and the model's configuration for the next step. In an extension of the experiment, we fine-tuned large variant of T5 (Raffel et al., 2020) and BART (Lewis et al., 2019) models. Furthermore, we employed the GPT-3.5 (Ouyang et al., 2022) model to perform the entity recognition task, exploring both zero-shot and few-shot learning setups. In order to accomplish the Named Entity Recognition (NER) task within the scope of few-shot learning, we curated a collection of 100 requirement statements. Notably, every entity appeared a minimum of five times throughout the entire collection. Subsequently, these instances are fed into the instruction prompt. In this scenario, the GPT-3.5 model achieved a accuracy of 92%. The performance metrics of all three models are presented in Table 7. Notably, the FiBER model demonstrates a unique advantage in comprehending software specification documents, attributed to its training on relevant requirements specification datasets. However, the GPT-3.5 model, when employed in a few-shot learning context, attains comparable performance to the FiBER model. Yet, the accessibility of the GPT-3.5 model is restricted, accompanied by notably greater expenses compared to the rest of the models.

(c) For recognition tasks, we use the different types of relationships among the different phrases and the root word of the sentence that is provided by the Component Extractor module.

(i) **Actor Recognition.** Our entity classification model is used to identify the actors by recognizing the *ACTOR* labels from the uNLR statements which have a subjective relationship (*nsubj*) with the root word. The extracted actors from three example uNLR sentences have been shown in Table 8.

(ii) **Resource Recognition.** Resources are recognized from the uNLR statements as those entities that have the object relationship (*obj* and *obl*) with the root word of the sentence. We do not consider *ORG*, *TIM* and *ACTOR* labels as possible *resources* of the intended goal model. Extracted resource metadata for three sample sentences have been presented in Table 9.

In Fig. 5, we can see how our classifier annotates our three sample sentences using the Entity Recognition Module.

Comparison of proposed classification model with existing approaches

In Table 10, we list some entity recognition systems that have been proposed for the software engineering domain. The SoftNER model (Tabassum et al., 2020) is a BERT-based solution tailored to excel in identifying code tokens and software-related labels in StackOverflow data. The entity recognition system proposed by of Ye et al. (2016) uses the 150 Stack Overflow posts to train with and recognizes 5 different classes of entities. The work achieves 78.17% of F1-Score on an average in recognizing these classes. Malik et al. (2021) trained their model on SRS documents of Dynamic Object-Oriented System (DOORS) (Hull et al., 2002) and classified 10 different classes with an F1-Score of 89% on an average. Another work (Herwanto et al., 2021) proposes an automated NER approach to detect privacy-related entities in user stories. Our proposed model is trained on PROMISE dataset and classifies 10 entity types of requirements engineering domain. Our transformer based classification model achieves 92% of F1-Score on average. Table 10 presents a comparative analysis among them.

4.3.2. Goal identification

The previous component extractor uses the Stanford dependency parser to extract the root and subtle information of a sentence like supporting verb to the main verb, auxiliary verbs, nominal compound and many more (Chen and Manning, 2014). These information are accepted by the *Goal Identification* module (marked as 3b in Fig. 2). The information are efficiently represented in a tree structure. In most of the cases, the *Goal Identification* module recognizes the main verb as the main process or *Goal*. The *Goal* can be combined with other words or compound words exhibiting the *obj*, *xcomp* or *amod* relationships with the main verb. However, in some situations where a main verb is not present in the sentence, the auxiliary verb, in combination with another word, acts as the root verb. For example, Fig. 6 shows such a case where no main verb is present in the sentence. The word *"available"* is acting as the root word. In such cases, the auxiliary verb is also combined to create the root word. In order to identify the *goal* from the uNLR statements, we completely rely on the dependency parser and we recognize the root of the sentence as the *goal*. Table 11 presents the list of goals that have been identified for three sample uNLR sentences.

4.3.3. Decomposition type prediction and subgoal identification

In this module (marked as 3c in Fig. 2), we have considered two types of goal decompositions - *AND* and *OR* - respectively. In order to detect the decomposition type from uNLR specifications in an autonomous manner, we perform supervised classification. We have prepared a dataset of 1200 natural language requirements statements and their classification labels. The requirements statements are collected

[8] https://github.com/tecoholic/ner-annotator.

[9] In software requirement context, BIO can be interpreted as *Beginning Inside* or *Outside* of the entity tag.

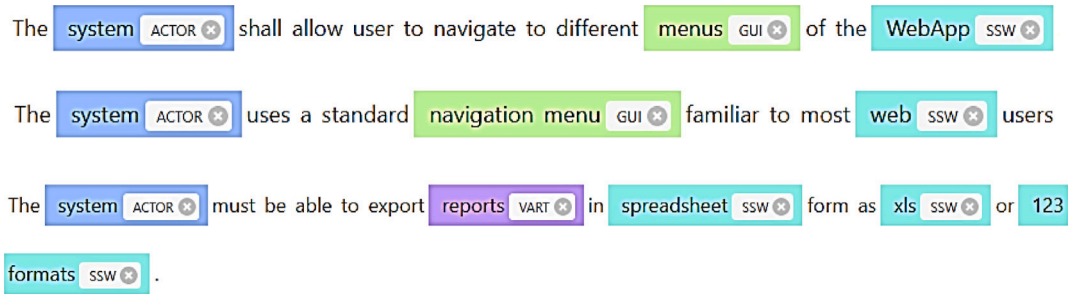


Fig. 5. Actor and resource identification using our entity recognition model.

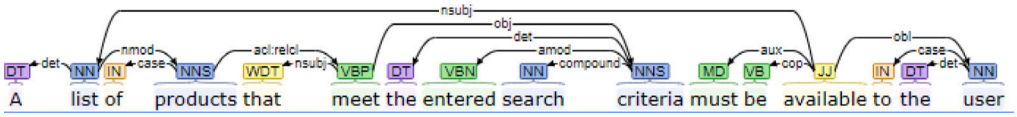


Fig. 6. Example of root word while main verb is absent.

Table 6
Example sentence annotated with entity labels in BIO format.

System	Shall	Have	MDI	Form	That	Allows	User	To	View	Graph	In	The	Screen
B-PLAT	O	O	B-GUI	I-GUI	O	O	B-ACTOR	O	O	B-VART	O	O	B-HW

Table 7
Performance of different entity recognition approaches.

Model	Precision	Recall	F1-Score	Accuracy
T5-large	89.65%	89.12%	88.88%	89.42%
BART-large	87.68%	87.42%	87.54%	87.42%
FiBER	92.34%	91.56%	91.95%	92.14%
GPT-3.5 Zero-shot	90.56%	89.24%	89.90%	89.42%
GPT-3.5 Few-shot	91.80%	90.68%	91.24%	92.18%

Table 8
Actor metadata.

Sentence	Metadata obtained
The system shall allow user to navigate to different menus of the WebApp	**Actor**: System
The system uses a standard navigation menu familiar to most web users	**Actor**: System
The system must be able to export reports in spreadsheet form as xls or 123 formats.	**Actor**: System

Table 9
Resource metadata.

Sentence	Metadata obtained
The system shall allow user to navigate to different menus of the WebApp	**Resource**: Menus
The system uses a standard navigation menu familiar to most web users	**Resource**: Navigation menu, Web
The system must be able to export reports in spreadsheet form as xls or 123 formats.	**Resource**: Reports, xls, 123 formats

from the PROMISE[10] and PURE[11] datasets. Among the 1200 uNLR statements, 587 are labeled as *OR* decompositions and 613 are labeled as *AND* decompositions. We use the *AdamW Optimizer* (Loshchilov and Hutter, 2017) with weight decay of 0.01 and learning rate of 5e−5 to tune up the FiBER model (Das et al., 2021) for 10 epochs. We then combine it with the *BinaryCrossEntropy* (Bruch et al., 2019) loss function since *Decomposition Type* detection is a two-class problem. (Please refer to Task 2 in Table 1.) The model achieves 79% of accuracy along with 78% of recall, 80% of precision and 79% of F1-Score. Table 12 shows the results for our three sample sentences.

Multiple interrelated goals and subgoals are often inherently present in uNLR specifications. It is a challenging task to identify the child subgoals of a particular parent goal. Once the *Decomposition Type* is predicted, we find the relationship between the root word and sub-phrases of the sentence that exhibit *obl* or *obj* relations. The idea is to identify the objects that can be represented as possible subgoals. On the other hand, it could also be the case that multiple phrases are connected with *conjunctions*. For example, *The system must be able to export reports in spreadsheet form as xls or 123 formats.* In such cases, we combine the root word with the phrases to generate the subgoals as shown in Table 13.

4.3.4. Softgoal identification

Softgoal identification from uNLR statements is also a crucial task. This module is marked as 3d in Fig. 2. In order to determine different softgoals involved in the system, we need to classify the uNLRs statements as functional or non-functional requirements (NFRs). We train the FiBER model with different parameters of the PROMISE_exp (Lima et al., 2019) dataset (Please refer to Task 3 in Table 1). The dataset consists of 969 requirements. The 969 requirements include 444 functional and 525 non-functional requirements. We consider total 12 classes for the classification task. There are 11 different NFR subclasses mentioned in the dataset and 1 is reserved for functional requirements. We use *AdamW Optimizer* during the training process. The *AdamW Optimizer* (Loshchilov and Hutter, 2017) considers a weight decay correction and does not compensate for bias. We use a weight decay of 0.01 and a maximal learning rate of 5e−05. The cross-entropy loss function is used for the training. The training of the model performed best for a batch size of 16 with 10 epochs. With .75-split of the dataset, the model achieves an accuracy of 91% on accuracy for multi-class classifier task. The model shows an average of 91% precision, and 90% for both recall and F1-Score. Furthermore, we fine-tuned the large variant of T5 (Raffel et al., 2020) and BART (Lewis et al., 2019) models on the PROMISE_exp dataset for requirements classification. we incorporated the GPT-3.5 (Ouyang et al., 2022) model to perform the classification task, covering both zero-shot and few-shot learning scenarios. In order

[10] http://promise.site.uottawa.ca/SERepository/datasets-page.html.
[11] https://zenodo.org/record/1414117#.Y1RPa3ZBy3A.

Table 10
Comparison of different entity recognition systems in software engineering domain.

Approach	Model	Dataset used	Recognized classes	Performance metric
Malik et al. (2021)	BERT, RoBERTa ALBERT	SRS documents of Dynamic Object Oriented Requirements System (DOORS) software	**O** - Non entity tokens **CORE** - Specific to software requirements domain **USER** - Specific user of the software **GUI** - Graphical user interface components **HARDWARE** - Computer hardware **LANGUAGE** - Programming language **API** - Software APIs **STANDARD** - Software/programming standards **PLATFORM** - Software platforms **ADJECTIVE** - Descriptive adjectives of software, hardware and design components **VERB** - Actions related to software, hardware and design components	Precision - 89% Recall - 90% F1-Score - 89%
Ye et al. (2016)	CRF + Brown Clustering	150 Stack Overflow posts	**O** - Non entity tokens **PL** - Programming language **Plat** - Platform **API** - API **Fram** - Tool library framework **Stan** - Software Standard	Precision - 82% Recall - 74% F1-Score - 78%
Tabassum et al. (2020)	BERT	150 Stack Overflow posts	**CLASS, VARIABLE, FUNCTION LIBRARY, VALUE, DATATYPE HTML XML TAG, APPLICATION, UI ELEMENT, LANGUAGE, FILE TYPE, DATA STRUCTURE, FILENAME, VERSION, DEVICE, OS, WEBSITE, and USER NAME**	Precision - 78% Recall - 79% F1-Score - 79%
Herwanto et al. (2021)	RoBERTa, BERT + BiLSTM + CRF	Manually labeled user stories	**Data Subject** **Processing** **Personal Data**	**Data Subject** F1 - 91% **Processing** F1 - 74% **Personal Data** F1%–72%
Proposed approach	BERT based FiBER Model	34 projects of PROMISE_exp dataset	**O** - Non entity tokens **VART** - Virtual Artifact **PART** - Physical Artifact **SSW** - Standard Software Resource **GUI** - Graphical components **HW** - Hardware Resource **PLAT** - Platform **EVE** - Event **TIM** - Time Indicator **ORG** - Organization **USER** - Specific user of the software	Precision - 92% Recall - 91% F1-Score - 92% Accuracy - 92%

Table 11
Goal metadata.

Sentence	Metadata obtained
The system shall allow user to navigate to different menus of the WebApp	**Goal**: Allow user to navigate
The system uses a standard navigation menu familiar to most web users	**Goal**: Use standard navigation menu
The system must be able to export reports in spreadsheet form as xls or 123 formats.	**Goal**: Export reports

Table 12
Decomposition type metadata.

Sentence	Metadata obtained
The system shall allow user to navigate to different menus of the WebApp	*NA*
The system uses a standard navigation menu familiar to most web users	*NA*
The system must be able to export reports in spreadsheet form as xls or 123 formats.	**DecompositionType**: OR

to conduct classification in a few-shot learning context, we curated 10 example requirement statements for each category, summing up to 110 examples embedded into the instruction prompt. In this few-shot learning scenario, the GPT-3.5 model achieved a noteworthy accuracy of 89%. The results, presented in Table 14, show that T5-large achieves highest performance and slightly better than the FiBER model. While our framework is flexible enough to use any requirements classification model, we chose to use the FiBER model because it is less than half the size of the T5-large model, which makes it more memory-efficient. Table 15 shows that the classification module has identified one NFR (*Usability*) out of three example sentences that we took for explaining the working process of the framework.

Comparison of proposed classification model with existing approaches

Amidst various approaches for NFR classification, Li et al. (2022) recently introduced DBGAT, an automatic requirements classification system that artfully combines BERT and Graph Attention Network (GAT). The method effectively utilizes dependency parse trees to capture implicit structure and syntactic features from requirements.

Table 13
Subgoal metadata.

Sentence	Metadata obtained
The system shall allow user to navigate to different menus of the WebApp	*NA*
The system uses a standard navigation menu familiar to most web users	*NA*
The system must be able to export reports in spreadsheet form as xls or 123 formats.	**Subgoal**: Export reports xls, Export reports 123 formats

Table 14
Requirements classification results of different transformer models.

Model	Precision	Recall	F1-Score	Accuracy
T5-large	94.42%	93.81%	94.11%	93.56%
BART-large	88.46%	87.85%	88.15%	88.28%
FiBER	91.16%	90.24%	90.70%	91.26%
GPT-3.5 Zero-shot	86.52%	85.82%	86.17%	86.21%
GPT-3.5 Few-shot	90.24%	89.12%	89.68%	89.46%

Table 15
Softgoal metadata.

Sentence	Metadata obtained
The system shall allow user to navigate to different menus of the WebApp	*NA*
The system uses a standard navigation menu familiar to most web users	**Softgoal**: Usability
The system must be able to export reports in spreadsheet form as xls or 123 formats.	*NA*

Table 16
State of the art for requirements classification task.

Author	Methodology	Evaluation metrics
Navarro-Almanza et al. (2017)	The approach is based on Convolutional Neural Network (CNN) to classify 12 categories of requirements	Precision: 81% Recall: 78.5% F1-Score: 77%
Dias Canedo and Cordeiro Mendes (2020)	The approach combines two text vectorization techniques with four ML algorithms to classify 12 categories of requirements.	F1-Score for binary classification: 91% F1-Score for 12 class classification: 78%
Kurtanović and Maalej (2017)	The approach uses SVM algorithm and involves features like meta-data, lexical, and syntactical information.	For binary classification F1-Score: 93% For multi-class classification F1-Score: range from 64% to 77%
Hey et al. (2020)	The work proposes NoRBERT that fine-tunes BERT for classification task using PROMISE NFR dataset.	F1-Score up to 92%
Li et al. (2022)	The work proposes DBGAT, that combines BERT and Graph Attention Network (GAT). Experiments has been conducted on PROMISE NFR dataset	F1-Score up to 88%
Proposed approach	We fine-tune FiBER model for classification task using PROMISE_exp dataset.	Precision: 91.167% Recall: 90.241% F1-Score: 90.702%

Our classification task closely resembles the work of Hey et al. (2020), where they successfully fine-tuned the BERT model for classification using the PROMISE dataset, achieving performance similar to ours. However, we made a distinctive choice by employing the FiBER model and fine-tuning it on the PROMISE_exp (Lima et al., 2019) dataset. The FiBER model's unique capability to understand and embed uNLR documents better than other transformer models motivated our selection for this classification task. Table 16 presents a comprehensive comparison of the performance metrics achieved by our proposed model with those of other existing approaches 16.

Table 17
Goal softgoal association metadata.

Sentence	Metadata obtained
The system shall allow user to navigate to different menus of the WebApp	*NA*
The system uses a standard navigation menu familiar to most web users	**Softgoal**: Usability; **AssociatedGoal**: Use standard navigation menu
The system must be able to export reports in spreadsheet form as xls or 123 formats.	*NA*

4.3.5. Goal-softgoal association

This is the last task of the goal model metadata generator module (marked as 3e in Fig. 2). It uses the goal model artifacts and relationships already identified in the previous tasks. We group all the requirements for each of the actors. Based on the output of the previous task, we identify all the non-functional requirement statements and the softgoals. We have also identified our goals and subgoals from the functional requirement statements in previous tasks. Finally, we measure the overall textual semantic similarity between the functional and non-functional requirements associated with the same actor. Additionally, we take into consideration the common (or similar) goals and resources among these sentences. If the method is unable to find reasonable similarity (similarity score ≥ 0.5) or goals in common, it simply returns NULL; otherwise, the goal-softgoal association is stored as goal model metadata. See Table 17.

4.3.6. Goal model metadata generation algorithm

Algorithm 1 explains the entire goal model metadata generation from uNLR specifications. It takes unstructured natural language requirements $\mathbb{NLR}$. The algorithm produces the goal model metadata $\mathbb{G}_M$ for the specification represented as $\mathbb{NLR}$. At the very first step the algorithm identifies all the actors from the given uNLR specifications and creates a set of actors $\mathbb{S}_A$ (line 2). Next, it repeats the entire procedure for each actor by iterating through all the actors in the actor set $\mathbb{S}_A$ (line 3). In the next step, requirements are grouped based on the specific actor $\mathbb{A}$ and stored in $\mathbb{R}_A$. In the next step, for each requirement R in $\mathbb{R}_A$, we extract goals and resources (line 5 to line 7). After extracting goals and resources, we find the association among goals and resources (line 8). G_R is the set contains goal that is associated with the corresponding resources. If cannot find any association G_R will be ϕ and iteration will take place for the next requirements from the set R_A. Otherwise, We keep the original requirement statements with the goal-resource association set as $G_{R'}$ (line 10). We include every such goal-resource association set along with requirements statements in another set G_{RA} (line 11). In the next step (line 14), we group up all the requirements having relationships among them in terms of goals and resources. SG_{RA} contains multiple sets of related requirements. We use FiBER model and cosine similarity to measure similarity among requirements. Common or similar goals and resources add an extra degree of accuracy for measuring relatedness among requirements. The next step is to predict the decomposition type (D_T) for each set of requirements (sg) in SG_{RA} (line 16 and 17). Goal decomposition information helps to identify the parent and sub goals for each set of related requirements (line 18 and 19). After extracting parent goal and subgoals, links among them are identified and stored in the goal model metadata $\mathbb{G}_M$ (line 20 and 21).

Algorithm 1 Goal Model Metadata Generation

Input:
a) $\mathbb{NLR}$: a set of natural language requirements tagged with POS and dependency relationships.

Output:
a) $\mathbb{G}_M$: the goal model metadata.

procedure GENERATE GOAL MODEL($\mathbb{NLR}$)
  $\mathbb{S}_A \leftarrow FindAllActors(\mathbb{NLR})$
  for $\mathbb{A} \in \mathbb{S}_A$ **do**
    $\mathbb{R}_A \leftarrow ExtractReqGroupByActor(\mathbb{A}, \mathbb{NLR})$
    for $R \in \mathbb{R}_A$ **do**
      $g \leftarrow ExtractGoal(R)$
      $r \leftarrow ExtractResource(R)$
      $G_R \leftarrow AssociateGoalResource(g, r)$
      if $G_R \neq \phi$ **then**
        $G_{R'} \leftarrow \{G_R, R\}$
        $G_{RA} \leftarrow G_{RA} \cup G_{R'}$
      end if
    end for
    $SG_{RA} \leftarrow GroupRelatedReq(G_{RA})$
    $\mathbb{G}_M \leftarrow \phi$
    for $sg \in SG_{RA}$ **do**
      $D_T \leftarrow PredictDecomposition(sg)$
      $G_P \leftarrow FindParent(sg)$
      $G_S \leftarrow FindSubGoal(sg)$
      $G_{M'} \leftarrow CreateParentLink(G_P, G_S, D_T)$
      $\mathbb{G}_M \leftarrow \mathbb{G}_M \cup G_{M'}$
    end for
  end for
end procedure

Table 18
Generated goal model metadata.

Sentence	Metadata
The system shall allow user to navigate to different menus of the WebApp	**Actor**: System; **Goal**: Allow user to navigate; **Resource**: Menus
The system uses a standard navigation menu familiar to most web users	**Actor**: System; **Softgoal**: Usability; **Associated Goal**: Use standard navigation menu; **Resource**: Navigation menu, Web
The system must be able to export reports in spreadsheet form as xls or 123 formats.	**Actor**: System; **Goal**: Export reports; **Subgoal**: Export reports xls, Export reports 123 formats; **DecompositionType**: OR; **Resource**: Reports, xls, 123 formats

4.4. tGRL model generator

The previous *Goal Model Metadata Generator* module generates the essential goal model metadata to generate the corresponding goal model specification. The *tGRL Model Generator* module of the framework (marked as 4 in Fig. 2) accepts the complete goal model metadata (as shown in Table 18) and generates tGRL descriptions of the goal model. Fig. 7 shows the generated tGRL representation for the example requirements statements. This tGRL description can be visualized as a graphical model using the open source jUCMNav (Abdelzad et al., 2015a) tool.

5. Experimental evaluation

We perform a crowdsourced experiment with our framework on 22 use cases in this section. We take the help of 335 crowdsource agents in our evaluation of the framework's performance. We look into the following three aspects in order to conduct a precise analysis of the applicability, effectiveness, and acceptance of our framework based on the observations from our experiment.

(i) *Acceptability of the goal models generated by our framework to Requirements Engineers.* A given uNLR specification can be represented by different goal model specifications depending upon the interpretations of the requirements engineer. The goal models generated by our framework might look different from other interpretations of experts. We aim to take feedback from the domain experts about the acceptability of the generated goal models in terms of correctness and completeness.
(ii) *Benefits of our approach.* The value proposition of our comprehensive framework is only limited by the benefits that goal models bring to requirement engineers and analysts. This is to observe whether requirements engineers are able to leverage the advantages of goal modeling analysis, both for legacy and modern systems.
(iii) *Feasibility in terms of time and space:* Requirement documents can be quite lengthy and it is a laborious job for the requirements engineer to analyze the whole document and extract a goal model which entails the specification. As a result, we investigate whether the framework operates within feasible time and space constraints, for generating the goal model.
(iv) *Scalability of the Framework:* Scalability stands as a fundamental aspect within this research and its practical applications, ensuring the solution's adaptability to handle the growing volumes of data and high-performing compatible deep learning models. This is to investigate that the design of the framework is flexible enough to accommodate the integration of more advanced models.

5.1. The crowdsourced experiment

We adopt a crowdsourcing based evaluation approach for our proposed framework. This approach has been used recently (Patwardhan et al., 2018; Arora et al., 2016) for estimating the acceptability of AI/ML-based solutions tailored for the requirements engineering community.

5.1.1. The questionnaire

The questionnaire which was circulated contains 22 uNLR use cases and their corresponding goal models generated by our proposed framework. Each use case consists of 3 to 4 unstructured or semi-structured sentences. The 22 use cases are deliberately manually composed by extracting statements from PROMISE and PURE datasets. The arrangement of these statements is intentionally maintained in an unstructured or, at the very least, semi-structured manner. This deliberate choice of unstructured sentences aims to test the framework's efficiency in handling diverse and challenging sentence structures. It is to be noted that, although our framework uses the tGRL language for representing the goal models, yet we present the i* visualization of the goal models in the questionnaire for provide better understanding and decision making. Fig. 8 shows an example of one such use case from the questionnaire. The crowdsource agents are given three options for providing their feedback on the correctness and completeness of the auto-generated goal models:

(i) *Correct*: According to the user, the artifacts represented in the goal model are correct and have an accuracy of more than 80% in capturing the uNLR statements.
(ii) *Partially Correct*: According to the user, some of the artifacts represented in the goal model are incorrect and have an accuracy between 50 – 80% in capturing the uNLR statements.
(iii) *Incorrect*: According to the user, the artifacts represented in the goal model are mostly incorrect and have an accuracy of less than 50% in capturing the uNLR statements.

```
grl{
    actor System{
        goal Allow_user_to_navigate{
            decompositionType: AND
            subgoal: null
            resource: Menus;
            softgoal: Usablility
        }
        goal Export_reports{
            decompositionType: OR
            subgoal: Export_reports_xls, Export_reports_123_formats
            resource: Reports, xls, 123_formats
        }
        goal Export_reports_xls;
        goal Export_reports_123_formats;
        softgoal Usability{
            resource: Navigation_menu, Web
        }
    }
}
```

Fig. 7. Generated tGRL representation of the three sample uNLR statements.

5.1.2. Specification of crowdworkers

We collected the feedback from 305 undergraduate students who were in the third year (fifth semester) of their B.Tech. degree program in Computer Science and Engineering.[12] and 30 postgraduate students who were pursuing their Masters in Computer and Information Science[13] The students were enrolled in Software Engineering courses of their respective institutions and were exposed to rigorous training of goal-oriented requirements engineering tools (like RE-Tools and jUCMNav) as part of their Software Engineering practical sessions. Each of students has developed more than 5–10 goal models as weekly assignments over a period of 8 weeks (2 months approximately) and also worked on a capstone project in groups of five members. Additionally, the students participated in live viva every week that involved interactions with the course instructor and three teaching assistants who are doing their Ph.D. in Requirements Engineering. Although the students (the crowdworkers) cannot be treated as experts, they can be expected to have an intermediate level of expertise in goal-oriented requirements engineering. The questionnaire was sent out as part of their course evaluation where they were asked to answer the questions with proper reasoning and justification. After submission, they were randomly asked to explain the rationale behind their feedback for specific use cases and marks were given based on the sincerity of effort put in by that student. This was done to prevent any sort of internal bias from effecting our evaluation process.

5.1.3. The results

We discuss the results of our crowdsource experiment with respect to the three aspects of evaluation mentioned at the beginning of the section.

(i) *Acceptability of the goal models generated.*
Fig. 9(a) shows the responses against all the 22 uNLR use cases, denoted as $R1$ to $R22$. It can be observed that the crowdworkers have accepted the generated goal models (except for $R6$, $R8$, $R11$, $R13$, and $R21$) as a correct representation of the given uNLR statements. Even among the five exceptions, only $R8$ has been classified as incorrect by majority of the crowdsource agents; the others have been accepted as partially correct. We also took the help of 10 requirements engineers who have been involved with the research community for at least three years. They were asked to provide their feedback on the questionnaire. Their opinion of the goal models validated the distribution of responses received from the crowdsource agents.
Furthermore, in Fig. 9(b), we observe that 77% of the total feedbacks received from the crowdsource agents conclude that the goal models produced by our framework (across the 22 use cases) are correct. On the other hand, 18% of the feedback acknowledged that the generated goal models are partially correct. Only 5% of all the feedback across all crowdsource agents and across all use cases suggested that the generated goal models were incorrect. Based on the results obtained from this crowdsourced experiment, we conclude that the goal models generated by the framework are well acceptable to a reasonable amount of people having certain amount of domain knowledge. With proper deployment and continuous enhancement of the framework, it could become a useful and widely acceptable open-source tool for the software engineering community.

(ii) *Benefits of our framework.*
With the aid of requirement goal models, the community of goal-oriented requirements engineering has offered frameworks, tools, and solutions for efficient management, analysis, and validation of system requirements. On the other hand, the majority of requirement specifications are documented in software industries using natural language statements, both in legacy requirement specification documents as well as agile story cards. Natural language specifications are prone to errors resulting from wrong interpretations, inconsistencies, and incompleteness. Our proposed solution makes it possible to derive requirement goal models from uNLR specifications, which would make it easier to carry out various types of requirements analyses like satisfiability analysis, entailment checking, inconsistency checking, and others. More specifically, the framework would ease the process for the requirements engineers to apply RE techniques and tools in order to manage volatile requirements.

(iii) *Feasibility in terms of time and space.*
In order to address this aspect, we analyze the time and space requirements of the framework with respect to the number of artifacts identified in the goal models.

[12] The B.Tech. students were from the Indian Institute of Information Technology, Vadodara, India.

[13] The Masters students were from the University of Calcutta, India.

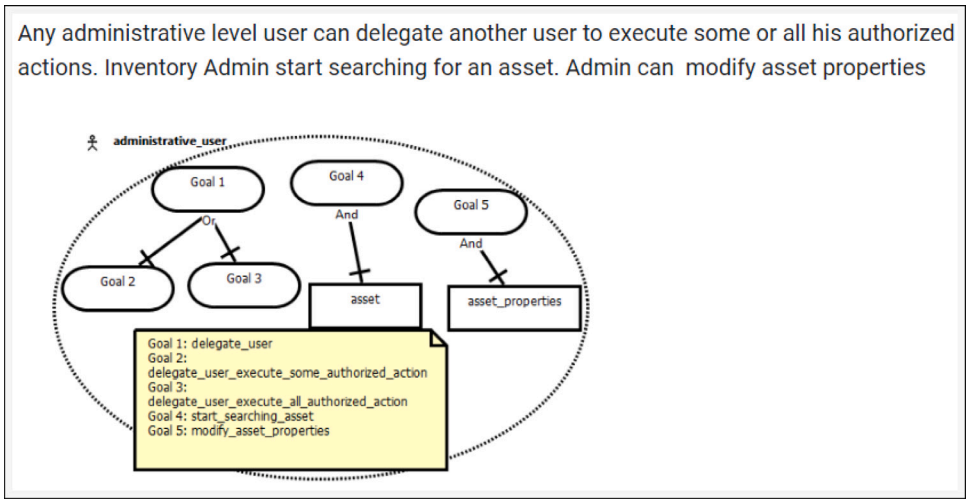
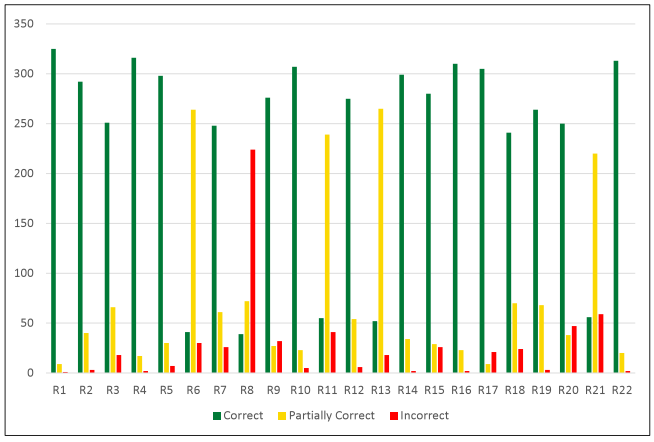
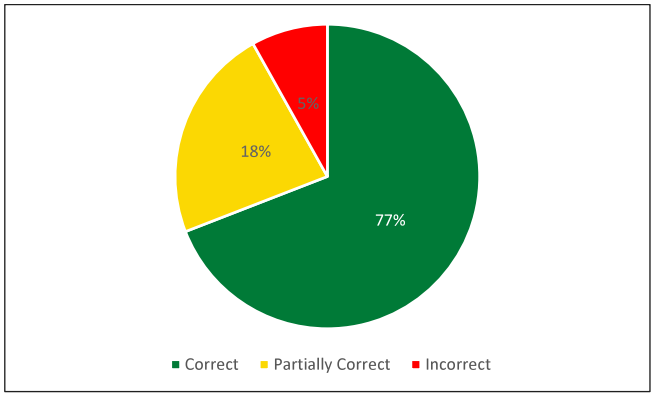


Fig. 8. Questionnaire of interview survey.



(a) Distribution of feedback for each of the 22 use cases.



(b) Overall distribution of feedback.

Fig. 9. Results of the crowdsourced experiment.

(a) **Time Analysis:** We consider a batch to consist the 22 use cases. We executed our approach 20 times on the same batch to get a good average estimate about the execution time. For each use case, the execution time of our framework for extracting the goal model from uNLR statements is in the order of minutes. Execution times were measured on a laptop with a 2.90 GHz CPU and 16 GB memory. The time requirements of each use case include the processing time of every module of the framework, starting from the processing of uNLR statements and ending with the generation of the goal model. The time required for each use case is analyzed in terms of a scatter plot and a linear regression line.

Fig. 10 shows the scatter plot along with the linear regression line. The linear regression equation has the form $Y = a + bX$, where Y is the dependent variable (time required to run the framework), X is the independent variable (number of components in the goal model), b is the slope of the line and a is the y-intercept. The equations for finding a and b are as follows:

$$a = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2}$$

$$b = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

Based on these equations, we determine the values of a and b as 65.22 and 4.96, respectively. We can observe that the execution time grows linearly as the number of generated goal model components increases. Thus, we can conclude that our framework takes reasonable amount of execution times without involving any GPU infrastructure and, hence, we expect our approach to scale to larger requirement documents efficiently.

(b) **Space Analysis:** We examine the space requirements of the different components of the framework in order to analyze the overall space requirements of the complete framework. The following list includes the major memory intensive components and their memory needs.

i. *Program variables and files:* 10 MB
ii. *Dependency Parser and POS Tagger*: 248 MB
iii. *FiBER model*: 1.7 GB
iv. *Entity recognition model*: 3.5 GB
v. *Softgoal Classification model*: 2.4 GB

Only the first two components are variables and depend on the size of the uNLR specification being worked upon. The memory requirements of the models (mentioned between iii - v) are invariant with respect to our framework. Over a certain period of time, the models may have to be retrained in order to take care of data drifts. The retraining activity will be sporadic in nature and does

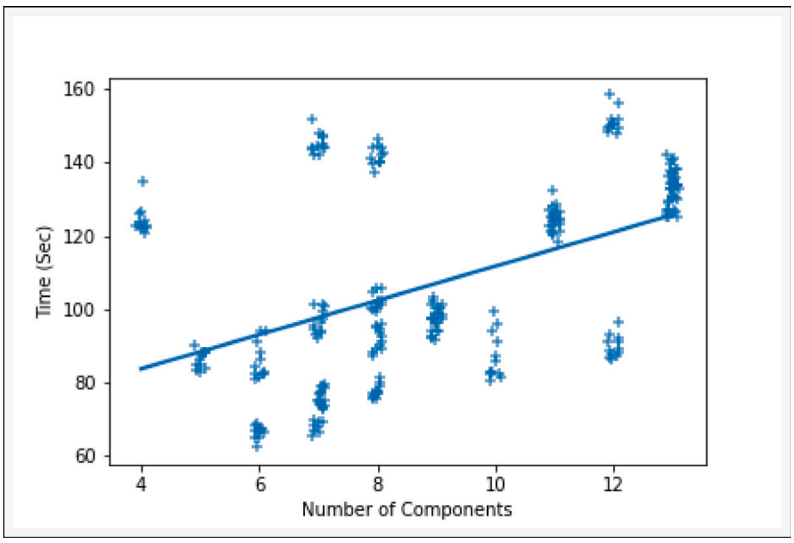


Fig. 10. Time required for extracting goal model components.

not affect our space and time analyses. Additionally, with recent advancements in computer systems and cloud infrastructures, we can expect our framework to use space efficiently.

(c) **Scalability of the Framework:** The framework harnesses the power of various NLP tasks, such as dependency parsing and POS tagging. Additionally, it utilizes the BERT-based FiBER model across different modules to achieve diverse objectives. The key lies in fine-tuning the FiBER model with task-specific datasets, tailoring it for various purposes. For example, the FiBER model is trained on an NER dataset to adeptly recognize *Actors* and *Resources*. Similarly, when fine-tuned on the PROMISE_exp dataset, it demonstrates its proficiency in classifying functional and non-functional requirements.

It is important to emphasize that this paper introduces an innovative framework designed to tackle the research problem holistically, without being constrained by the performance of any individual module. As time progresses, the framework remains adaptable, enabling the replacement of deep learning models with even more advanced and compatible ones, thereby ensuring scalability and future-readiness.

5.2. Application in popular case studies

In this section, we present the results of applying our framework on two popular case studies explored in the goal-oriented requirements engineering literature. These include the meeting scheduling system (Yu, 2001) and an online shopping system (Giorgini et al., 2005). We have carefully selected the use cases of Meeting Scheduling System and Online Shopping System for this study. These use cases have been considered by several other research studies (Yu et al., 2008; Liaskos et al.) because they closely resemble real-world scenarios. We manually created the requirements specification statements for both of the use cases. The requirements statements are intentionally kept unstructured. Fig. 12 shows such examples of requirements statements for the meeting scheduling system use case. The meeting scheduling use case and the online shopping system use case contain 46 and 52 natural language requirements statements respectively. Furthermore,

Table 19
Properties of the use case goal models.

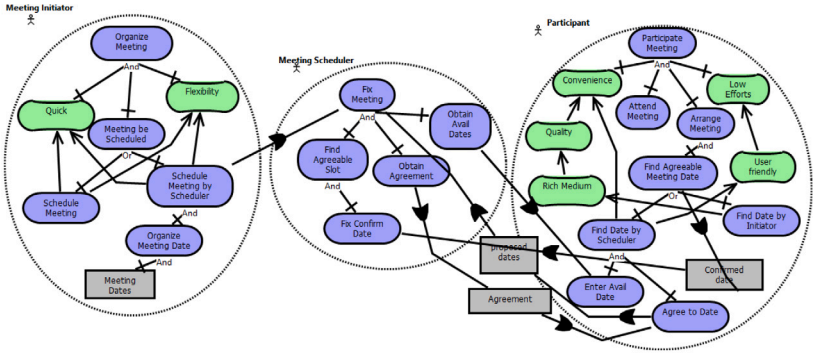
Properties	Use cases	
	Meeting scheduling system	Online shopping system
Number of goals	18	23
Number of goal decompositions	20	22
Number of resources	4	7
Number of goal-resource associations	7	7
Number of SoftGoals	7	2
Number of goal-Softgoal associations	10	2
Number of natural language requirements statements	46	52

Table 19 presents different properties of these goal models. A part of the meeting scheduling system is presented in Fig. 11(a). Fig. 11(b) shows the corresponding goal model derived by our framework from uNLR specifications of the meeting scheduler. This figure highlights in red those components which could not be identified and extracted by our framework.

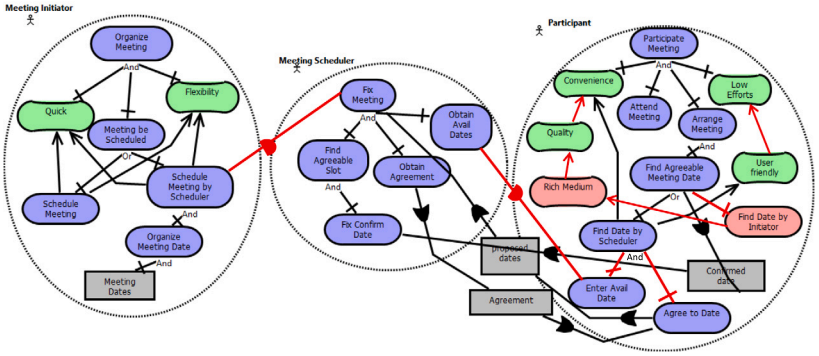
Our evaluation of the generated goal model is measured based on six characteristics:

1. Number of Goals.
2. Number of Goal Decompositions.
3. Number of Resources.
4. Number of Goal-Resource Associations.
5. Number of Softgoals.
6. Number of Goal-Softgoal Associations.

We evaluate our framework on both case studies and present the results in Table 20. In the case of meeting scheduling system, our framework recognizes 17 goals among 18 available goals and gives some false positives. Thus it achieves 95.83% of accuracy. There are 20 goal decompositions that have been presented in the original goal model. Our algorithm shows 87.5% of accuracy by recognizing 17 goal decompositions along with some false positives. On identifying resources and their associations with goals, our framework recognizes 6 softgoals out of 7 available softgoals. Thus, it shows 91.66% of accuracy but suffers from false positives. The original meeting scheduling goal



(a) Goal Model for Meeting Scheduling System



(b) Generated Goal Model for Meeting Scheduling System. The elements in red could not be identified by our framework.

Fig. 11. Comparative analysis of original goal model and generated goal model.

- Meeting initiator can schedule meeting by scheduler.
- Meeting initiator must organize date during Scheduling meeting by scheduler.
- Participants can find agreeable meeting dates to arrange meeting.
- Participant can find agreeable date by using scheduler.
- To find meeting date by meeting scheduler, participant must enter available dates.

Fig. 12. Example of requirements statements for meeting scheduling system.

model has 10 goal-softgoal associations and out of which our framework recognizes 6 associations. Thus, it achieves 60% of accuracy. The detailed analysis of precision, recall and F1-Score have been presented in Table 20.

In the online shopping case study, our framework recognizes 21 goals among 23 available goals with few false positives. Thus, it achieves 88.46% of accuracy. In the original goal model, there are 22 goal decompositions and our framework shows 92.30% of accuracy by recognizing 20 goal decompositions. There are 7 resources available in the online shopping goal model and our framework recognizes all of them but suffers from some false positives. It presents an accuracy of 92.59%. Similar results can be seen in the case of goal and resource associations. The framework recognizes all the soft goals and their associations present in the goal model. The detailed analysis of precision, recall, and F1-Score have also been presented in Table 20.

6. Threats to validity

In this section, we list the limitations and other threats to the validity of our framework.

- *Human Feedback Bias*: The use of human feedback to evaluate the applicability of the framework may introduce bias in the evaluation data. This bias could impact the measurement of the actual capability and applicability of the framework, especially when

Table 20
Experimental results of applying our proposed framework.

Parameters	Use cases and results				
Goal identification	Number of Successful Goal Identification	Precision	Recall	F1-Score	Accuracy
	Meeting Scheduling System				
	17	1	0.94	0.97	0.95
	Online Shopping System				
	21	0.95	0.91	0.93	0.88
Goal decomposition	Number of Successful Goal Decompositions Identification	Precision	Recall	F1-Score	Accuracy
	Meeting Scheduling System				
	17	1	0.85	0.91	0.87
	Online Shopping System				
	20	1	0.91	0.95	0.92
Resource identification	Number of Successful Resource Identification	Precision	Recall	F1-Score	Accuracy
	Meeting Scheduling System				
	4	0.66	1	0.80	0.91
	Online Shopping System				
	7	0.77	1	0.87	0.92
Goal-Resource association	Number of Successful Goal-Resource Association	Precision	Recall	F1-Score	Accuracy
	Meeting Scheduling System				
	7	0.77	1	0.87	0.92
	Online Shopping System				
	7	0.77	1	0.87	0.92
Softgoal identification	Number of Successful Softgoal Identification	Precision	Recall	F1-Score	Accuracy
	Meeting Scheduling System				
	6	0.85	0.85	0.85	0.91
	Online Shopping System				
	2	1	1	1	1
Goal-Softgoal association	Number of Successful Goal-Softgoal Association	Precision	Recall	F1-Score	Accuracy
	Meeting Scheduling System				
	6	0.85	0.60	0.70	0.80
	Online Shopping System				
	2	0.66	1	0.80	0.96

the feedback is provided by students with limited professional experience.

- *Unable to capture some Goal Model artifacts:* Our framework is not mature enough to capture some concepts of goal models such as dependencies, contribution values on contribution links, and softgoal interdependency graphs.
- *Difficulty in finding relatedness among uNLRs:* Requirements statements that are inter-related may span across multiple statements. It is still difficult for our framework to find the relation between two goals (or softgoals) that reside across multiple statements or even across a paragraph.
- *Difficulty in identifying goal decompositions:* Our framework can identify goal decompositions from a single sentence or within two consecutive sentences. However, it may be the case where goal decomposition for a particular goal is specified in a span of multiple statements. In such a case, our framework will fail to identify the goal decompositions. Also, we observed that our framework fails to identify goal decompositions when the subgoals are not well mentioned in the uNLR statements.
- *Significant Time Requirements:* As our framework employs several NLP tasks and algorithms, it needs to load large language models. The framework requires a reasonable amount of time, specifically to accomplish classification tasks using language models and the measurement of semantic similarity.

Examples of scenarios where our framework faces limitations are showcased in Table 21.

7. Conclusion

In this paper, we propose a comprehensive framework that supports the construction of goal model specifications from unstructured natural language requirement specifications. Our approach seamlessly integrates diverse NLP algorithms, techniques, and advanced transformer models to achieve this goal. Notably, the framework's adaptability ensures future scalability, allowing the replacement of current deep learning models with more advanced and compatible alternatives as they emerge over time. During the development of the framework, we come up with a requirements classification framework that outperforms state-of-the-art requirements classification models. We also developed an entity type recognition system in this domain which can work independently outside the framework. We have performed an extensive crowdsource experiment to analyze the acceptability of the goal models generated by our framework. The acceptability analysis has been further validated by taking the opinion of experienced requirements engineers. In the future, we aim to capture softgoal interdependency graphs and the contribution values from the uNLR specifications in order to make the generated goal models more complete. We also intend to perform empirical evaluations of our framework on some industry use cases in real-life projects. We also plan to test our framework

Table 21
Examples of such cases where our framework fails.

Example sentence	Problem	Possible cause for the problem
Any administrative level user can delegate another user to execute some or all his authorized actions	Unable to identify exact subgoal "some authorized actions"	The proposed method is unable to find the phrase, acting as subgoal after the word "some".
Any administrative level user or inventory user can edit an asset that belongs to its department; same thing for faculty user	Unable to recognize same goal for the faculty user.	The second sentence is referring to a phrase of the previous sentence and the proposed mechanism is unable to recognize it.
The system will allow the admin or lecturer to plan meetings	Irrelevant subgoals "allow admin to plan meetings" and "allow lecturer to plan meetings" recognized.	Structural diversity of the language.
The users are able to send and receive messages.They can confirm their availability	Unable to recognize actor in the second sentence.	Anaphora resolution has not been done.
Registered and Guest user both can view the products.	Unable to associate goal "view products" to Guest user.	Proposed approach assumes only one actor should be specified in a single requirements statement.
Any authorized user can made creating request to borrow an asset	Unable to attach goal to all users having authorization.	Proposed approach cannot find relevant actors referred by the words like "any", "all", "every", "each" and so on.

for agile platforms and look into the challenges in capturing similar concepts from user stories.

CRediT authorship contribution statement

Souvick Das: Conceptualization, Methodology, Software, Writing – original draft, Data curation. **Novarun Deb:** Conceptualization, Investigation, Data curation, Writing – review & editing, Validation. **Agostino Cortesi:** Conceptualization, Supervision, Writing – review & editing, Project administration, Funding acquisition. **Nabendu Chaki:** Conceptualization, Supervision, Writing – review & editing, Project administration.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Souvick Das reports financial support was provided by Ca' Foscari University Department of Environmental Sciences Informatics and Statistics. Souvick Das reports financial support was provided by iNEST-Interconnected NordEst Innovation Ecosystem.

Data availability

https://github.com/svk-cu-nlp/NLR-to-Goal-Model.

Acknowledgments

Work partially supported by SERICS (PE00000014) and iNEST (ECS 00000043) projects funded by PNRR NextGeneration EU.

References

Abdelzad, V., Amyot, D., Alwidian, S.A., Lethbridge, T., 2015a. A textual syntax with tool support for the goal-oriented requirement language. In: IStar. pp. 61–66.

Abdelzad, V., Amyot, D., Lethbridge, T.C., 2015b. Adding a textual syntax to an existing graphical modeling language: experience report with grl. In: International SDL Forum. Springer, pp. 159–174.

Agarwal, O., Subramanian, S., Nenkova, A., Roth, D., 2019. Evaluation of named entity coreference. In: Proceedings of the Second Workshop on Computational Models of Reference, Anaphora and Coreference. Association for Computational Linguistics, Minneapolis, USA, pp. 1–7. http://dx.doi.org/10.18653/v1/W19-2801, URL https://aclanthology.org/W19-2801.

Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., 2015. Automated checking of conformance to requirements templates using natural language processing. IEEE Trans. Softw. Eng. 41, 944–968.

Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., 2016. Extracting domain models from natural-language requirements: approach and industrial evaluation. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 250–260.

Ben Abdessalem Karaa, W., Ben Azzouz, Z., Singh, A., Dey, N., Ashour, A.S., Ben Ghazala, H., 2016. Automatic Builder of Class Diagram (Abcd): An Application of Uml Generation from Functional Requirements, vol. 46, Wiley Online Library, pp. 1443–1458.

Bhatia, J., Breaux, T.D., Schaub, F., 2016. Mining Privacy Goals from Privacy Policies using Hybridized Task Recomposition, vol. 25, ACM New York, NY, USA, pp. 1–24.

Bragilovski, M., Dalpiaz, F., Sturm, A., 2022. Guided derivation of conceptual models from user stories: A controlled experiment. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 131–147.

Bruch, S., Wang, X., Bendersky, M., Najork, M., 2019. An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. In: Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval. pp. 75–78.

Casagrande, E., Woldeamlak, S., Woon, W.L., Zeineldin, H.H., Svetinovic, D., 2014. Nlp-Kaos for Systems Goal Elicitation: Smart Metering System Case Study, vol. 40, IEEE, pp. 941–956.

Cheikhi, L., Abran, A., 2013. Promise and isbsg software engineering data repositories: A survey. In: 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement. IEEE, pp. 17–24.

Chen, D., Manning, C.D., 2014. A fast and accurate dependency parser using neural networks. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. EMNLP, pp. 740–750.

Chong, E.K., Zak, S.H., 2013. An Introduction to Optimization, vol. 75, John Wiley & Sons.

Cleland-Huang, J., Settimi, R., Zou, X., Solc, P., 2006. The detection and classification of non-functional requirements with application to early aspects. In: 14th IEEE International Requirements Engineering Conference. RE'06, IEEE, pp. 39–48.

Cowie, J., Lehnert, W., 1996. Information Extraction, vol. 39, ACM New York, NY, USA, pp. 80–91.

Dalpiaz, F., Van Der Schalk, I., Brinkkemper, S., Aydemir, F.B., Lucassen, G., 2019. Detecting terminological ambiguity in user stories: Tool and experimentation. Inf. Softw. Technol. 110, 3–16.

Darimont, R., Van Lamsweerde, A., 1996. Formal Refinement Patterns for Goal-Driven Requirements Elaboration, vol. 21, ACM New York, NY, USA, pp. 179–190.

Das, S., Deb, N., Cortesi, A., Chaki, N., 2021. Sentence Embedding Models for Similarity Detection of Software Requirements, vol. 2, Springer, pp. 1–11.

Deeptimahanti, D.K., Babar, M.A., 2009a. An automated tool for generating uml models from natural language requirements. In: 2009 IEEE/ACM International Conference on Automated Software Engineering. IEEE, pp. 680–682.

Deeptimahanti, D.K., Babar, M.A., 2009b. An automated tool for generating uml models from natural language requirements. In: 2009 IEEE/ACM International Conference on Automated Software Engineering. IEEE, pp. 680–682.

Deeptimahanti, D.K., Sanyal, R., 2011. Semi-automatic generation of uml models from natural language requirements. In: Proceedings of the 4th India Software Engineering Conference. pp. 165–174.

Deshpande, G., Sheikhi, B., Chakka, S., Zotegouon, D.L., Masahati, M.N., Ruhe, G., 2021. Is bert the new silver bullet?-an empirical investigation of requirements dependency classification. In: 2021 IEEE 29th International Requirements Engineering Conference Workshops. REW, IEEE, pp. 136–145.

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Dias Canedo, E., Cordeiro Mendes, B., 2020. Software requirements classification using machine learning algorithms. Entropy 22, 1057.

Dozat, T., Qi, P., Manning, C.D., 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. In: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. pp. 20–30.

Duran, M.B., Pina, A.N., Mussbacher, G., 2015. Evaluation of reusable concern-oriented goal models. In: 2015 IEEE International Model-Driven Requirements Engineering Workshop (MoDRE). IEEE, pp. 1–10.

Ferrari, A., Spagnolo, G.O., Gnesi, S., 2017. Pure: A dataset of public requirements documents. In: 2017 IEEE 25th International Requirements Engineering Conference. RE, IEEE, pp. 502–505.

Giorgini, P., Mylopoulos, J., Sebastiani, R., 2005. Goal-oriented requirements analysis and reasoning in the tropos methodology. Eng. Appl. Artif. Intell. 18, 159–171.

Güneş, T., Aydemir, F.B., 2020. Automated goal model extraction from user stories using nlp. In: 2020 IEEE 28th International Requirements Engineering Conference. RE, IEEE, pp. 382–387.

Herwanto, G.B., Quirchmayr, G., Tjoa, A.M., 2021. A named entity recognition based approach for privacy requirements engineering. In: 2021 IEEE 29th International Requirements Engineering Conference Workshops. REW, IEEE, pp. 406–411.

Hey, T., Keim, J., Koziolek, A., Tichy, W.F., 2020. Norbert: Transfer learning for requirements classification. In: 2020 IEEE 28th International Requirements Engineering Conference. RE, IEEE, pp. 169–179.

Hull, E., Jackson, K., Dick, J., 2002. Doors: a tool to manage requirements. In: Requirements Engineering. Springer, pp. 187–204.

Ibrahim, M., Ahmad, R., 2010. Class diagram extraction from textual requirements using natural language processing (nlp) techniques. In: 2010 Second International Conference on Computer Research and Development. IEEE, pp. 200–204.

Jackson, M., 1995. Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices. ACM Press/Addison-Wesley Publishing Co.

Kochbati, T., Li, S., Gérard, S., Mraidha, C., 2021. From user stories to models: A machine learning empowered automation.. MODELSWARD 10, 0010197800280040.

Kof, L., 2005. Natural language processing: Mature enough for requirements documents analysis? In: International Conference on Application of Natural Language to Information Systems. Springer, pp. 91–102.

Kumar, D.D., Sanyal, R., 2008. Static uml model generator from analysis of requirements (sugar). In: 2008 Advanced Software Engineering and its Applications. IEEE, pp. 77–84.

Kurtanović, Z., Maalej, W., 2017. Automatically classifying functional and non-functional requirements using supervised machine learning. In: 2017 IEEE 25th International Requirements Engineering Conference. RE, Ieee, pp. 490–495.

Landhäußer, M., Körner, S.J., Tichy, W.F., 2014. From Requirements to Uml Models and Back: How Automatic Processing of Text Can Support Requirements Engineering, vol. 22, Springer, pp. 121–149.

Letsholo, K.J., Zhao, L., Chioasca, E.-V., 2013. Tram: A tool for transforming textual requirements into analysis models. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 738–741.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L., 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461.

Li, G., Zheng, C., Li, M., Wang, H., 2022. Automatic requirements classification based on graph attention network. IEEE Access 10, 30080–30090.

Liaskos, S., Khan, S.M., Litoiu, M., Jungblut, M.D., Rogozhkin, V., Mylopoulos, J., Behavioral Adaptation of Information Systems Through Goal Models, vol. 37, Elsevier, 2012, pp. 767–783.

Liaskos, S., McIlraith, S.A., Sohrabi, S., Mylopoulos, J., 2010. Integrating preferences into goal models for requirements engineering. In: 2010 18th IEEE International Requirements Engineering Conference. IEEE, pp. 135–144.

Lima, M., Valle, V., Costa, E., Lira, F., Gadelha, B., 2019. Software engineering repositories: expanding the promise database. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering. pp. 427–436.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

Loshchilov, I., Hutter, F., 2017. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101.

Lucassen, G., Dalpiaz, F., van der Werf, J.M.E., Brinkkemper, S., 2016. Improving agile requirements: the quality user story framework and tool. Requir. Eng. 21, 383–403.

Malik, G., Cevik, M., Bera, S., Yildirim, S., Parikh, D., Basar, A., 2021. Software requirement-specific entity extraction using transformer models.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. pp. 3111–3119.

More, P., Phalnikar, R., 2012. Generating Uml diagrams from natural language specifications, Vol. 1, pp. 19–23.

Mussbacher, G., Amyot, D., Araújo, J., Moreira, A., Weiss, M., 2007. Visualizing aspect-oriented goal models with aogrl. In: Second International Workshop on Requirements Engineering Visualization (REV 2007). IEEE, p. 1.

Navarro-Almanza, R., Juarez-Ramirez, R., Licea, G., 2017. Towards supporting software engineering using deep learning: A case of software requirements classification. In: 2017 5th International Conference in Software Engineering Research and Innovation. CONISOFT, IEEE, pp. 116–120.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al., 2022. Training language models to follow instructions with human feedback. arXiv preprint arXiv:2203.02155.

Patwardhan, M., Sainani, A., Sharma, R., Karande, S., Ghaisas, S., 2018. Towards automating disambiguation of regulations: using the wisdom of crowds. In: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 850–855.

Popescu, D., Rugaber, S., Medvidovic, N., Berry, D.M., 2007. Reducing ambiguities in requirements specifications via automatically created object-oriented models. In: Monterey Workshop. Springer, pp. 103–124.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J., et al., 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. 21, 1–67.

Robeer, M., Lucassen, G., Van der Werf, J.M., Dalpiaz, F., Brinkkemper, S., 2016. Automated extraction of conceptual models from user stories via nlp. pp. 196–205. http://dx.doi.org/10.1109/RE.2016.40.

Rupp, C., Simon, M., Hocker, F., 2009. Requirements Engineering und Management, vol. 46, Springer, pp. 94–103.

Sagar, V.B.R.V., Abirami, S., 2014. Conceptual Modeling of Natural Language Functional Requirements, vol. 88, Elsevier, pp. 25–41.

Sanh, V., Debut, L., Chaumond, J., Wolf, T., 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

Santorini, B., 1990. Part-of-Speech Tagging Guidelines for the Penn Treebank Project. University of Pennsylvania, School of Engineering and Applied Science ....

Shimada, H., Nakagawa, H., Tsuchiya, T., 2017. Constructing a goal model from requirements descriptions based on extraction rules. In: Asia Pacific Requirements Engeneering Conference. Springer, pp. 175–188.

Spijkman, T., Molenaar, S., Dalpiaz, F., Brinkkemper, S., 2021. Alignment and granularity of requirements and architecture in agile development: A functional perspective. Inf. Softw. Technol. 133, 106535.

Straka, M., Straková, J., 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. pp. 88–99.

Tabassum, J., Maddela, M., Xu, W., Ritter, A., 2020. Code and named entity recognition in stackoverflow. arXiv preprint arXiv:2005.01634.

Van Lamsweerde, A., 2001a. Goal-oriented requirements engineering: A guided tour. In: Proceedings Fifth Ieee International Symposium on Requirements Engineering. IEEE, pp. 249–262.

Van Lamsweerde, A., 2001b. Goal-oriented requirements engineering: A guided tour. In: Proceedings Fifth Ieee International Symposium on Requirements Engineering. IEEE, pp. 249–262.

Van Lamsweerde, A., 2009. Requirements Engineering: From System Goals to UML Models to Software, vol. 10, John Wiley & Sons, Chichester, UK.

Weber-Jahnke, J.H., Onabajo, A., 2009. Mining and analysing security goal models in health information systems. In: 2009 ICSE Workshop on Software Engineering in Health Care. IEEE, pp. 42–52.

Ye, D., Xing, Z., Foo, C.Y., Ang, Z.Q., Li, J., Kapre, N., 2016. Software-specific named entity recognition in software engineering social content. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering. SANER, vol. 1, IEEE, pp. 90–101.

Yu, E., 2001. Agent-oriented modelling: software versus the world. In: International Workshop on Agent-Oriented Software Engineering. Springer, pp. 206–225.

Yu, E., 2011. Modeling strategic relationships for process reengineering. Soc. Model. Requir. Eng. 11, 66–87.

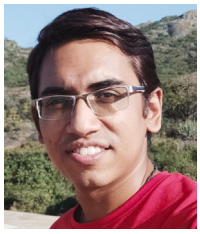
Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J., Leite, J.C., 2008. From goals to high-variability software design. In: Foundations of Intelligent Systems: 17th International Symposium, ISMIS 2008 Toronto, Canada, May (2008) 20-23 Proceedings. Vol. 17, Springer, pp. 1–16.

Zeman, D., Hajic, J., Popel, M., Potthast, M., Straka, M., Ginter, F., Nivre, J., Petrov, S., 2018. Conll 2018 shared task: Multilingual parsing from raw text to universal dependencies. In: Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. pp. 1–21.

Zhou, Q., Li, T., Wang, Y., 2022. Assisting in requirements goal modeling: a hybrid approach based on machine learning and logical reasoning. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems. pp. 199–209.

Zhou, N., Zhou, X., 2004. Automatic acquisition of linguistic patterns for conceptual modeling.

Souvick Das is a Research Associate at the Department of Environmental Science, Informatics, and Statistics, of Ca' Foscari University, Venice, Italy and is currently pursuing Ph.D. in Computer Science and Engineering from University of Calcutta, India. He has received B.Sc and M.Sc in Computer Science from West Bengal State University, India in the year 2012 and 2014 respectively. He has qualified UGC NET-JRF in the year of 2016.



Novarun Deb is an Assistant Professor at the Indian Institute of Information Technology, Vadodara, India. Dr. Deb was a Research Associate at the Department of Environmental Science, Informatics, and Statistics, of Ca' Foscari University, Venice, Italy, for more than two years. He has done his Masters and Ph.D. in Requirements Engineering from the Department of Computer Science and Engineering, University of Calcutta.

Nabendu Chaki is a Professor in the University of Calcutta, Kolkata, India. He is sharing 7 international patents including 4 US patents. Dr. Chaki has authored 7 books and nearly 200 Scopus Indexed papers in Journals and International conferences. He is the founder Chair of ACM Professional Chapter in Kolkata. He was active in 2009–2015 for developing international standards in Software Engineering and Service Science as a Global (GD) member for ISO-IEC.

Agostino Cortesi is a Professor of computer science in Ca' Foscari University, Venice, Italy. He has extensive experience in the area of static analysis and software verification techniques, with particular emphasis on security applications. He published more than 150 papers in high level international journals and proceedings of international conferences. Currently, he serves as co-Editor in Chief of the book series "Services and Business Process Reengineering" published by Springer Nature.