The Use of the Pre-Trained BERT and GPT-3 Models to Automate the Composing of Use Case Descriptions

Ayad Tareq Imam[1] and Iyad Altawaiha[2]

[1]Isra University
[2]Universiti Putra Malaysia (UPM

August 17, 2023

THE USE OF THE PRE-TRAINED BERT AND GPT-3 MODELS TO AUTOMATE THE COMPOSING OF USE CASE DESCRIPTIONS

Ayad Tareq Imam

Dept. of SE, DS & AI; Faculty of Information Technology;
Isra University; Amman; Jordan
Alzobaydi_ayad@iu.edu.jo

Iyad Altawaiha

Dept. of SE and IS, Faculty of Computer Science and Information Technology,
Universiti Putra Malaysia (UPM), Serdang 43400, Malaysia.
Iyad.twh@yahoo.com

ABSTRACT

Composing the use case description, which comes in the form of a textual table, is an essential and critical task while developing software. Usually, this process results from manually extracting the actors and their use cases and defining other essential information like the flow of events for each use case. This paper proposes a new Intelligent Computer Aided Software Engineering (I-CASE) tool to automatically generate use case descriptions and use case diagrams from user requirements by utilisation of the pre-trained GPT-3 and BERT Large Language Model (LLM).
The evaluation showed the tool's ability to outperform previous studies in creating use case descriptions based on high accuracy in extracting actors and their use cases from text written in natural English.
While there is a lack of test cases, especially in natural languages other than English, this is the first use of the Large Language Model (LLM) to automate the generation of use case descriptions and draw the use case diagrams.

KEYWORDS

BERT; GPT-3; Use Case Description; NLP; SRL; I-CASE.

1. Introduction

Effective communication between clients and software analysts is crucial for successful software development as misunderstandings resulting from ambiguous language can lead to poor software functionality and project failure [1]. Collecting complete and precise requirements is an essential but challenging task in software engineering (SE), with errors often remaining concealed until later stages of the Software Development Life Cycle (SDLC) [2]. The precise definition of software requirements by using a systematic modelling approach of system specifications from a functional (use case) perspective can help software development teams to create software that meets the needs of all stakeholders, including end-users and clients, and prevent any misunderstandings that could lead to project failure [3].

Use case descriptions are crucial to software engineering, particularly in object-oriented software requirements engineering. The development of these descriptions entails identifying actors and their corresponding use cases, which are the external entities interacting with the system, such as humans, systems, companies, or devices, and the system behaviours [4]. The process of identifying actors and use cases produces the requirement specification document that serves as a foundation for subsequent software development phases, such as design, testing, and validation [5]. Each use case description typically consists of several key elements, including the use case name, actors, preconditions, postconditions, and flow of events [6].

The use case name provides a brief, descriptive title that reflects the goal of the interaction. Actors represent the external entities participating in the use case, while preconditions specify any conditions that must be met before initiating the use case. Postconditions outline the expected state of the system after the successful completion of the use case. The flow of events details the sequence of steps and

interactions between the system and the actors, leading to the desired outcome [6]. An example of the use case description is shown in Fig 1.

Element	Description
Use Case Name	Log In
Actors	User
Preconditions	User has a valid account and is on the login page.
Postconditions	User is authenticated and redirected to the homepage, or an error message is displayed if authentication fails.
Flow of Events	1. User navigates to the login page 2. User enters their username and password. 3. User clicks the "Log In" button. 4. System validates the credentials 5. If credentials are valid, the user is redirected to the homepage 6. If credentials are invalid, an error message is displayed, and the user is prompted to re-enter their credentials.

Fig 1: Use Case Description Example

In most software development projects, the extracting of use cases is generally done manually by system analysts, who use the user requirements to develop use case models [7]. However, as software development becomes increasingly complex, automating the process of requirements engineering becomes a growing need, to create comprehensive and more accurate functional requirements specifications while speeding up the process as well. Automated tools are needed that can parse user requirement documents, automate extracting actors, and use cases. Extracting use cases is a form of extracting information from text, which is automated using the increasingly prevalent Natural Language Processing (NLP) approach [8].

This paper aims to automate the extraction of actors and use cases from user requirements and the generation of use case descriptions. To achieve this, we propose a tool that utilizes the pre-trained Bidirectional Encoder Representation from the Transformers (BERT), and the Generative Pre-trained Transformer 3 (GPT-3) to extract actors and use cases and to generate the use case description.

BERT and GPT are two of the most influential NLP models developed in recent years. Both models are based on transformer architecture and have made significant contributions to the field of NLP by achieving state-of-the-art results on a wide range of tasks [9].

BERT is a deep learning model that was developed by Google and was first introduced in 2018 by Devlin et al. [10]. Unlike previous NLP models, BERT [11] is bidirectional, meaning it can consider a word's pre- and post-context in each sequence. This allows a better understanding of the meaning of words and sentences and produces more accurate predictions. BERT is highly effective in a variety of NLP tasks, such as sentiment analysis, NER, question answering, sentence classification, and SRL [11] [12].

GPT, on the other hand, is a generative model that can generate new high-quality natural language text that is like the text it was trained on [13]. OpenAI introduced GPT in 2018, and since then it has achieved state-of-the-art results on a variety of NLP tasks, such as language modelling, text completion, and machine translation [14]. Unlike BERT, GPT is not bidirectional, but it still deeply understands language structure and can generate coherent and meaningful text [15]. Both BERT and GPT are pre-trained on large corpora of text data, which allows them to learn the underlying structure of language and develop a deep understanding of natural language. This pre-training is then fine-tuned on specific NLP tasks, making them highly effective for a wide range of applications [9].

The research approach used in this paper starts with the study of related works and passes through proposing a tool, testing, and reporting the results of the proposed tool, evaluating the proposed tool, and ending with a conclusion.

2. Related Works

Previous research studies employed several methods to automatically extract actors and use cases from the natural language requirements, one of which involved utilizing business models. In one such approach, Dijkman et al. [16] presented an approach that derived functional requirements specifications (use case diagram) from business process models. The meta-models were established for both business process models & use case diagrams and compared to derive a formal mapping that serves as the foundation of their approach.

Another approach was proposed by Giachetti et al. [17], which utilized a model-based digital engineering process that involves recording all program data in machine-readable models. The aim is to replace the extensive written requirement specifications traditionally generated during new systems' development.

NLP is a vital tool for comprehending the text of software requirements. It leverages techniques like text analytics and text mining to perform this task. Many researchers used NLP tools to extract actors and cases. For example, Hyder et al. [18] proposed an approach called LESSA based on NLP. This approach involves conducting lexical and syntactic analysis to generate use case diagrams from software requirement text. Similarly, Deeptimahanti et al. [19] presented an approach that utilizes NLP tools to create use case diagrams from user requirements. Their approach is designed to break down complex requirements into simpler ones, streamlining the process of generating use case diagrams.

Part of Speech (POS) tagging is a subfield of NLP that involves classifying words in a text as lexical terms. This technique utilizes NLP methods to identify the POS of each word, such as noun, verb, adjective, etc., in a sentence. Alksasbeh [20] presented an approach that leverages POS to create use-case diagrams. The proposed method identifies POS from sentences and applies specific grammatical patterns to label them, thereby enabling the identification of actors and use cases. Following this, a stemming algorithm and heuristic rules are applied to specify actors and use cases and subsequently generate the use case diagram. Likewise, Arman et al. [21] used Stanford Parser to parse user requirements. The method extracts a list of verbs and nouns from the text, which can then be used as actors and use cases in generating use case diagrams.

Imam et al. [22] introduced a novel approach that combines NLP with ANN to improve the software requirement elicitation process. They used an NLP parser to produce lexicons, syntaxes, and dependencies. The combination of ANN and NLP in their approach improved the accuracy of the POS tagging and allowed for better identification of actors and use cases. Similarly, Jebril et al. [7] developed an approach utilizing the thematic role principle to extract use cases and actors. This technique is a linguistic domain used to analyse the text from a semantic level. They automated the process of extracting the actors and actions utilizing thematic roles. Imam et al. [23] proposed an approach to extract the actors and use cases from user requirements by training Support Vector Machine (SVM). They utilized POS tagging and Name Entity Recognition (NER) to label the different words in the software requirements text and SVM to classify them in the user functional requirement text.

However, these approaches have several limitations, including difficulty in identifying complex sentence structures, inability to capture the context of the text, and limited semantic understanding [24] [8] [25]. Moreover, traditional techniques, such as lexical and syntactic analysis, have been used, but they may not be sufficient for complex tasks such as generating use case descriptions.

As a conclusion from these previously reported related works, modern pre-trained adapter models such as BERT and GPT-3, which have emerged as game-changers in NLP [26], have not yet been used to achieve use case description configuration and extract actors and their use cases. In this study, a pre-trained GPT-3 model (to generate a use case description) and a BERT model (to accomplish the SRL process) are used to develop a new Intelligent Computer Aided Software Engineering (I-CASE) tool [27]. This tool aims to overcome the limitations of existing approaches used in generating use case descriptions and extracting the actors and their use cases as will be explained in the following sections.

3. The Proposed Use-Case Description Generator Based on Transformer Models

The proposed Use-Case Description Generator Based on Transformer Models (UCDGT) tool has been developed using the Python programming language and the PyQt5 library. The purpose of the tool is to extract actors and use cases from a given text and generate both the use case diagram and the use case description.

As shown in Fig 2, the UCDGT tool performs several steps to analyze a given text. First, it tokenizes the text and identifies each token's POS tags. The tool extracts the modal verbs from the POS tags. Then, the tool loads an SRL prediction model from Allennlp and applies it to the text. Using the NLTK library, the tool extracts the base form of the verbs and their associated arguments from the SRL prediction. Next, the tool iterates through the extracted verbs and retrieves their corresponding arguments. The tool skips any modal verbs and uses the remaining verbs to generate use cases and actors. Then, it generates the use case diagram from the extracted actors and use cases. Finally, the tool leverages OpenAI's GPT-3 to generate a description for each resulting use case.

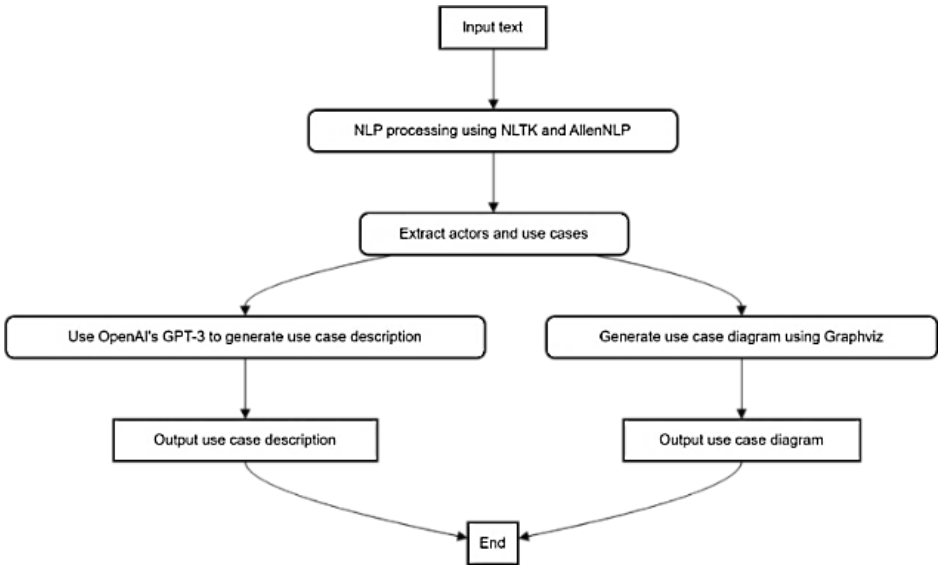

Fig 2: Flow Diagram of the UCDGT Tool

The tool has been designed as a graphical user interface (GUI) application, enabling users to input text, extract relevant information, and view the results. The tool includes two tabs; the first tab is used to extract actors, use cases, and generate the use case diagram and descriptions, while the second tab is used to extract the SRL. Fig 3 provides snapshots of the tool's interface for SRL Tab and use case Extraction Tab, respectively.

Fig 3. Snapshot of SRL Tab and Snapshot of use case Extraction Tab

As mentioned above, the UCDGT tool utilizes various NLP libraries, including NLTK and AllenNLP, to perform text processing and analysis. Additionally, it incorporates OpenAI's GPT-3 to generate a description for the extracted use cases. To draw the resulting use case diagram, Graphviz, a library for generating graphs and diagrams, is utilized. The libraries used in the UCDGT tool are:

- PyQt5: This is a library for creating desktop applications with a GUI. This library is used to create the GUI for our tool.
- Natural Language Toolkit (NLTK): This library is used to tokenize the sentence and identify the part-of-speech tags. In our tool, it is used to extract the modal verbs to be excluded.
- Allennlp: This library for NLP provides pre-trained models for various NLP tasks. It is used to load the structured prediction SRL model and parse the text.
- Openai: This library provides access to OpenAI's artificial intelligence models and tools. This library is used to generate the use case description for the resulting use cases using GPT-3.
- Graphviz: A library for generating graphs and diagrams; it is used to draw the use case diagram.

The UCDGT tool combines a pre-trained BERT model for SRL (AllenNLP-SRL) and GPT-3 (Text-Davinci-003) to extract actors, use cases, and the use case description from a given text. The tool first tokenizes the input sentence, identifies the part-of-speech tags for each token, and extracts the modal verbs from the tags. It then uses a pre-trained SRL model to parse the sentence and extract the base form of the verb and its arguments. The code then filters out model verbs and uses the remaining verbs to construct use cases and actors in addition to generate a description for the extracted use cases using GPT-3.

3.1 Extracting Actors and Use Cases

In this step, we use an SRL model to extract actors and use cases from given sentences. First, the SRL model is loaded, and the sentence is passed through the model to extract the base form of the verb and its arguments. Then, the arguments are processed for each extracted verb, and the actors are stored separately from the use cases. The arguments are categorized into two types - ARG0 and ARG1. ARG0 represents the actor or the entity that acts, and the use case is formed by combining the base verb and the argument represented by (ARG1). Fig 4 illustrates the pseudocode used for extracting actors and use cases from a sentence.

```
Load the SRL model
Use the model to parse the sentence
Extract the base form of the verb and the arguments from the prediction
For each verb in the prediction:
    Get the base form of the verb
    If the verb is a model verb, skip it
    Extract the arguments for the current verb being processed
        For each argument:
                If it is an (ARG0), store it in the actors list
                If it is an (ARG1), combine it with the base verb and store it in the use cases list
Print out the use cases and actors lists
```

Fig 3: The Pseudocode for Extracting Actors and Use-Cases

3.2 Generating the Use Case Description

This step involves utilizing OpenAI's GPT-3 model to generate the use case description for each extracted use case identified in the previous step. Fig 5 illustrates the pseudocode for generating a use case description for each extracted use case. A carefully crafted prompt is created, instructing the GPT-3 model to produce a use case description based on the extracted use case. Several parameters need to be specified to ensure the GPT-3 model generates the desired output. The following is a brief explanation of the parameters used in the GPT-3 model for developing use case descriptions:

- Prompt: The input prompt or query for the model to generate a response to. In our study, the prompt is formulated to request a use case description for each use case extracted in the previous step.
- Engine: The specific GPT-3 model to use. In this study, the text-DaVinci-003 engine is used, which is the most powerful and expensive one offered by OpenAI.
- Max_tokens: The maximum number of tokens (words or symbols) allowed in the generated output. This parameter is used to limit the length of the generated description.
- n: The number of completions to generate. This parameter determines the number of alternative descriptions generated by the model.
- Stop A string or list of strings that cause the model to stop generating further text when encountered in the generated output. In this study, none is used to allow the model to generate the full description without stopping prematurely.
- Temperature: A parameter that controls the randomness or creativity of the generated output. A lower temperature value leads to more conservative and predictable text, while a higher value can lead to more imaginative and varied text. A value of 0.5 is used in this study, which is a relatively moderate level of randomness.

```
For each use case in extracted use cases:
        prompt= "generate the use case description for" + {output_of_step 1}
        completion = openai.Completion.create (
        engine = "Text-Davinci-003",
        prompt = prompt
        Set max_tokens to 2048
        Set n to 1
        Set stop to None
        Set temperature to 0.5
        )
    Print the generated description
```

Fig 4: Pseudocode to Generate Use Case Description

Once the prompt and parameters have been configured, the GPT-3 model is executed, generating a use case description for the extracted use case. This description then fills the corresponding fields in the use case description table. The process is automatically repeated for each extracted use case, creating a comprehensive set of use case descriptions derived from the initial text input.

3.3 Generating Use Case Diagram

In this step, the process of generating the use case diagram begins by initializing a new Graphviz graph object, which serves as the foundation for building the use case diagram. The primary loop iterates through the list of verbs and their base forms, extracted earlier in the process. For each verb, the code evaluates whether it is a modal verb. If it is, the loop skips the current verb and proceeds to the next one in the sequence.

When encountering a non-modal verb, the code extracts the corresponding arguments (ARG0 and ARG1) from the verb's tags and prediction words. ARG0 typically represents the actor, while ARG1 signifies the object involved in the use case. If the verb does not have an ARG0, it is skipped, and the loop continues with the following verb.

The use case is generated by concatenating the base verb and the ARG1, while the actors are extracted from ARG0. The results are then appended to the 'results' list, which will store the extracted use cases and actors for further processing.

For each extracted use case and its corresponding actors, the code creates nodes in the Graphviz graph object. An edge is added between the actor node and the use case node, representing their relationship in the diagram.

Upon completion of the loop, the code generates a use case diagram by creating nodes for the actors and use cases and edges to represent their relationships. The Graphviz graph object is then rendered and displayed as an image, visually representing the extracted use cases and actors from the input text. Fig 6 illustrates the pseudocode for generating the use case diagram.

```
initialize Graphviz graph object

for each verb and base_verb in extracted verbs and base_verbs:
    if verb is a modal verb or "be" or "have":
        continue

    extract ARG0 and ARG1 from the verb's tags and prediction words
    if verb has no ARG0:
        continue

    generate use_case by concatenating base_verb and ARG1
    extract actors from ARG0
    append use_case and actors to results list

    create nodes for use_case and actors in the Graphviz graph object
    add edge between actor node and use_case node in the graph

render and display or save the Graphviz graph object as a use case diagram
```

Fig 5: Pseudocode to Generate Use Case Diagram

4. Testing and Results

This step aims to assess the UCDGT system and report the results gained from this testing. The testing was accomplished on the two functions of the UCDGT system, which are the extracting of the use case and actors, and the generating of the use case description.

4.1 The Experiments on Extracting Actors and Use Cases

To evaluate the classification performance of the tool, we conducted testing on four case studies covering a variety of industries and use cases. Each case study was manually reviewed and analyzed to extract the actors, and use cases, which were then compared with the results generated by the tool. The following provides an overview of each case study's testing and result.

4.1.1 Case Study 1- A Food Delivery Application

The description of the food delivery application is shown in Fig 7. The UCDGT tool successfully extracted all the actors and use cases in this case study very closely to the results of manual extraction, with only minor differences in structure. Table 1 shows the results obtained from the UCDGT tools and those of manual extraction.

"The customer can easily create an account by their email or phone number. Then, the customer browses the menu page with the items and desirable times and can quickly place an order for their favorite meal online. The customer can cancel the order only within a specific period. The customer can also view the status of the order. The customer picks up the order once it is ready. Cook can view customer's order, confirm the order, and send notifications if the order is ready. Cook can also edit order status. The Administrator can modify the menu of food items. Also, the Administrator can edit the menu information, such as price and items available presently. The Administrator can view the Order queue and reassign the order." [29]

Fig 6: The Case Study 1- Food Delivery

Table 1. Tool-Extraction and Manually Extraction of Actos and Use-Cases from the Description of Food Delivery Application

Actor	Tool-Extracted Use Cases	Manual- Extracted Use Cases
Customer	Create an account	Create an account
	Browse menu page	Browse menu page
	Place an order for your favourite meal	Place an order online
	Cancel order	Cancel the order within a specific period
	View the status of an order	View the status of the order
	Pick order	Pick up the order
Cook	View the customer's order	View the customer's order
	Confirm order	Confirm the order
	Send notifications	Send notifications
	Edit order status	Edit the order status
Administrator	Modify the menu of food items	Modify the menu of food items
	Edit the menu information, such as price and items available presently	Edit the menu information
	View order queue	View the order queue
	Reassign order	Reassign the order

4.1.2 Case Study 2 – The ATM System

The description of the ATM system is shown in Fig 8. The UCDGT tool was able to extract actors and use cases with one error. However, the overall results were still considered useful for further analysis and development. Table 2 reports the detailed results.

"The bank client must be able to deposit an amount to and withdraw an amount from his or her accounts using the bank application. Each transaction must be recorded, and the client must have the ability to review all transactions performed against a given account. Recorded transactions must include the date, time, transaction type, amount and account balance after the transaction. A bank client can have two types of accounts: a checking-account and a saving-account. For each checking account, one related saving-account can exist. The application must verify that a client can gain access to his or her account by identification via a personal identification number (PIN) code. Neither a checking account nor a saving account can have a negative balance. The application should automatically withdraw funds from a related saving account if the requested withdrawal amount on the checking-account is more than its current balance. If the saving-account balance is insufficient to cover the requested withdrawal amount, the application should inform the user and terminate the transaction". [30]

Fig 7: Case Study 2 – The ATM System

Table 2. Tool-Extraction and Manually Extraction of Actos and Use cases from the Description of the ATM System

Actor	Tool-Extracted Use Cases	Manual- Extracted Use Cases
Bank client	Deposit an amount	Deposit an amount
	Withdraw an amount	Withdraw an amount
	Use bank application	Review all transactions
	Review all transactions performed against a given account	Use bank application
	Have two types of accounts: a checking account and a saving-account	Have two types of accounts: a checking account and a saving-account
	gain access to his or her account	gain access to his or her account using a PIN
Application	verify that a client can gain access to his or her account identification via a personal identification number (PIN) code	verify that a client can gain access to his or her account using a PIN code
	withdraw funds	withdraw funds
	inform user	inform user

	terminate transaction	terminate transaction
Account	check	

It should be notable that the word "application" in this description is considered an actor based on the definition of the actor, which could be defined as any person, group, or system that interacts with the software system being developed, which can also include software applications. In many cases, an application may be an actor that interacts with the system. In this case study, the "application" is an external system that interacts with the system being developed (ATM system).

4.1.3 Case Study 3 - The Cafeteria Ordering System (COS)

As shown in Fig 9, the description of this system uses uncommon terms that are ambiguous and uncertain, even to human readers, to denote actions or responses that occur within the system and that are the services provided by the system to the user. The ambiguous and uncertain terms are considered a true challenge to the UCDGT tool as the problems of ambiguity and uncertainty are the default challenges in the processing of the natural language. However, the tool accurately extracted all actors and use cases. The tool's output matches the manual extraction with slight wording and structure differences as shown in Table 3.

"The web application shall let a Patron, who is logged into the COS, place an order for one or more meals. The web application shall confirm that the Patron is registered for payroll deduction to place an order. If the Patron is not registered for payroll deduction, the web application shall give the Patron options to register now and continue placing an order, to place an order for pickup in the cafeteria, or to exit from the COS. The web application shall prompt the Patron for the meal date. If the meal date is the current date and the current time is after the order cutoff time, the web application shall inform the patron that it's too late to place an order for today. The Patron may either change the meal date or cancel the order. The Patron shall specify whether the order is to be picked up or delivered. If the order is to be delivered and there are still available delivery times for the meal date, the Patron shall provide a valid delivery location". [31]

Fig 8: Case Study 3 – COS

Table 3. Tool-Extraction and Manually Extraction of Actos and Use cases from the Description of the COS System

Actor	Tool-Extracted Use Cases	Manual- Extracted Use Cases
Web Application	let a patron place an order for one or more meals	let a patron place an order for one or more meals
	confirm that the patron is registered for payroll deduction to place an order	confirm that the patron is registered for payroll deduction to place an order
	give options to register now and continue placing an order	give options to register now and continue placing an order
	prompt the Patron	prompt the Patron for the meal date
	inform patron	inform the patron that it is too late to order
Patron	log in to cos	Log in to cos
	place an order for one or more meals	place an order for one or more meals
	register	Register into cos
	continue placing an order	continue placing an order
	exit from cos	exit from cos
	change meal date	change meal date
	Cancel order	Cancel order
	specify whether the order is to be picked up or delivered	specify whether the order is to be picked up or delivered
	provide a valid delivery location	provide a valid delivery location

Worth to note that the term "web application" (rather than just "application" word) was used in the description of the COS as an actor to denote an external system that interacts with the COS.

4.1.4 Case Study 4 – The Call for Help Application

The Call for Help Application, which is illustrated in Fig 10, features multiple actors and their use cases. The UCDGT tool successfully identified the relevant actors and use cases in this description with one encountered error, which makes the output useful for future development. Table 4 reports a detailed breakdown of the results.

"Users can register, log in, select the services, views the schedule on which time the service provider is available, reserve the most suitable slot, and then receive the service. After receiving the service user gives feedback. Location Service Provider (LSP) authenticates user, check the availability of technicians in that particular location which the user entered. Admin governs the overall working of the system, and can verify the customer and the service provider, manages the services category, views all the pending services, and lastly, views the feedback if the feedback is given by the user. Technician can Login and get notified about their next destination by LSP. Technician views their schedule and the location provided by the user seeking help. Then technician provides services". [29]

Fig 9: Case Study 4 – The Call for Help Application

Table 4. Tool-Extraction and Manually Extraction of Actos and Use cases from the Description of the Call for Help Application

Actor	Tool-Extracted Use Cases	Manual- Extracted Use Cases
User	Register	Register
	Log	Log
	Select services	Select services
	View Schedule	View Schedule
	reserve the most suitable slot	reserve the most suitable slot
	receive service	receive service
	enter that particular location	enter particular location
	give feedback	Give feedback
	seek help	
Location Service Provider	check the availability of technicians in that particular location which the user entered	check the availability of technicians in that particular location which the user entered
	authenticate user	authenticate user
	notify technician	notify technician
Admin	govern the overall working of the system	govern the overall working of the system
	verify customer and service provider	verify customer and service provider
	manage services category	manage services category
	view all pending services	view all pending services
	View feedback	View feedback
Technician	Login	Login
	View schedule and location provided	View schedule and location provided
	provide services	provide services

4.2 The Experiment on Generating the Use Case Description

The UCDGT tool was also assessed to generate the description for different use cases. The tool was able to generate the use case description, and the resulting description closely matched the expected description for each use case, demonstrating the tool's ability to generate the use description. Fig 11 shows a snapshot of the use case description generated by the UCDGT tool for the following text "*Users can register, log in, select the services, view the schedule on which time the service provider is available, reserve the most suitable slot, and then receive the service*".

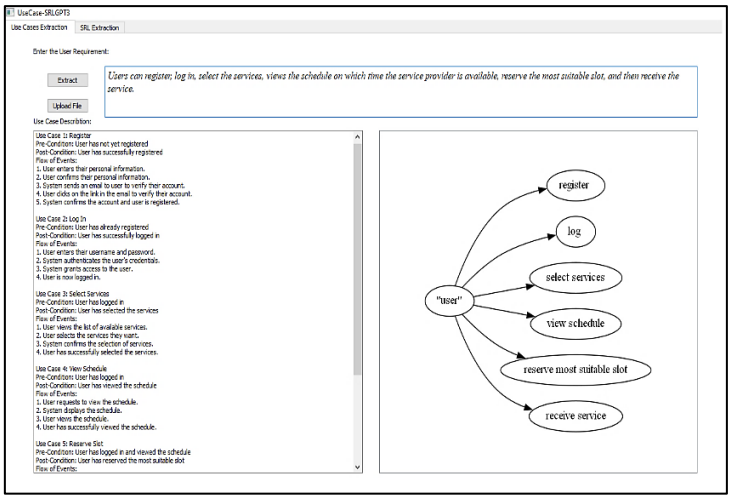

Fig 10: A Snapshot of the Use Case Description Generated by the Tool

4.3 The Experiments on Generating the Use Case Diagram

The capability of the UCDGT tool to generate use case diagrams was evaluated. Fig 12 provides an example of a use case diagram produced by the UCDGT tool based on the following input text: “The customer can easily create an account by their email or phone number. Then, the customer browses the menu page with the items and desirable times and can quickly place an order. The customer can cancel the order only within a specific period. The customer can also view the status of the order. Cook can view customers’ orders, confirm the order, and send notifications if the order is ready". The tool demonstrated the ability to create use case diagrams accurately representing the extracted information.

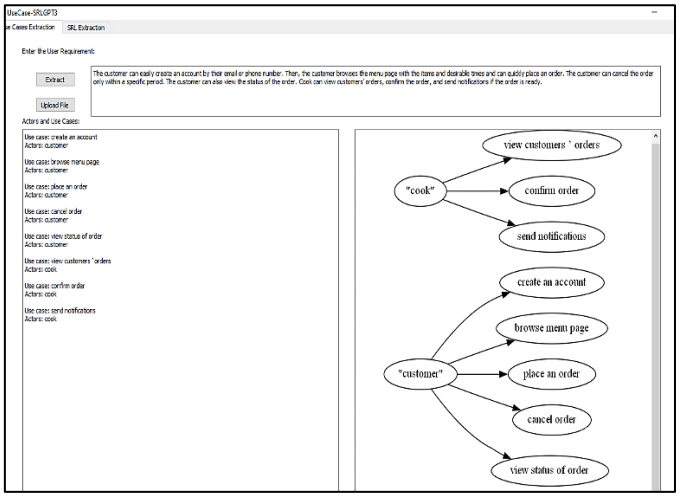

Fig 11: A Snapshot of the Use Case Diagram Generated by the Tool

5. Evaluation

This section aims to demonstrate the effectiveness of our tool in extracting actors and use cases from user requirement specifications. To measure the accuracy and completeness of the classification, we used the binomial classification accuracy metric, commonly used to evaluate the performance of classification models [28]. We also measured the tool's precision, recall, and F-measure for both actors and use cases.

The precision metric evaluates the tool's accuracy in extracting valid actors and use cases, while the recall metric measures the completeness of the extraction process. The F-measure is the harmonic mean of precision and recall, providing an overall measure of the tool's effectiveness in extracting actors and use cases. To calculate these metrics, we used the following formulas [28]:

- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- F-measure = $\frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$

where True Positive (TP) is the number of valid extracted actors and use cases, False Positive (FP) is the number of invalid extracted actors and use cases, and False Negative (FN) is the number of actors and use cases not extracted.

Table 5 shows the number of TP, FP, and FN for both actors and use cases in each case study. Table 6 shows the precision, recall, and F-measure for both actors and use cases in each case study and the overall average across all case studies.

Table 5. Number of TP, FP, and FN for Actors and Use Cases in each Case Study

Case Study	Actors					
	TP	**FP**	**FN**	**TP**	**FP**	**FN**
Food Delivery Application	3	0	0	14	0	0
ATM system	2	1	0	10	1	0
Cafeteria Ordering System	2	0	0	14	0	0
Call for Help Application	4	0	0	19	1	0

As shown in Table 5, our tool extracted many valid actors and use cases with few false positives and false negatives across all case studies. The precision, recall, and F-measure results in Table 6 demonstrate that the tool performed well in terms of both accuracy and completeness, with an overall average precision of 91.67% for actors and 96.23% for use cases. These results demonstrate the effectiveness of our tool in extracting actors and use cases from user requirement specifications.

Table 6. Precision, Recall, and F-measure for Actors and Use Cases

Case Study	No. of Words	Actors					
		Precision %	**Recall %**	**F-measure %**	**Precision %**	**Recall %**	**F-measure %**
Food Delivery Application	125	100	100	100	100	100	100
ATM system	168	66.67	100	80	90.91	100	95
Cafeteria Ordering System	176	100	100	100	100	100	100
Call for Help Application	123	100	100	100	95	100	97.44
Average		91.67	100	95.00	96.23	100	98.17

To further evaluate the effectiveness of our tool, we compared its performance to that of three previous studies that also focused on extracting actors and use cases from user requirement specifications. We summarized the results in Table 7, which shows each study's precision, recall, and F-measure values. Our tool outperformed two studies in all three metrics, indicating a higher accuracy and completeness in recognizing the actors and use cases in the input documents. For the third study, our tool achieved higher recall and F-measure scores but slightly lower precision than the previous approach. These results demonstrate the superiority of our tool in extracting actors and use cases from user requirements.

Table 7. Comparison of Metrics for Tool Performance with Related Studies

Related Studies	Precision %	Recall %	F-measure %
Bajaj et al. [29]	97.4	91.95	94.44
Alksasbeh [20]	84	96	89.6
Imam et al. [22]	76	76.2	76.09
Proposed Tool	**93.95**	**100**	**96.59**

Our tool distinguishes itself from the previous studies by being the first to generate a description for each extracted use case. This feature provides additional benefits to the tool's users, such as a clearer

understanding of the interactions between actors and the system and a more precise representation of the use cases.

6. Conclusion

This research proposes an I-CASE tool to automate the composing of the use case description next to the automatic extracting of the actors and their use cases from user requirements by using NLP and AI tools and techniques. The proposed UCDGT tool utilizes pre-trained models; including GPT-3 model (to generate a use case description) and the BERT model (to achieve the SRL process) from OpenAI. The accuracy of the UCDGT tool to extract the actors and their use cases is evaluated on four case studies from different industries. The evaluation of the results, which is achieved by using the binomial classification accuracy metric, demonstrates that the proposed UCDGT tool is highly accurate in extracting actors and use cases, with only minor structural differences compared to manual extraction. To evaluate the effectiveness of the UCDGT tool, a comparison of its performance to three previous studies is made, in which the comparison focused on extracting actors and use cases from user requirement specifications. The comparison demonstrated the originality of the UCDGT tool in composing the use case description, and the superiority of the UCDGT tool in extracting actors and use cases from user requirements. In short, this study differs from the conventional approaches by:

- The use of generative AI, particularly with the advent of OpenAI's GPT-3, which is the most extensive Artificial Neural Network (ANN) that can answer questions, summarize lengthy texts, write paragraphs, translate languages, and generate programming code.
- The use of SRL, an NLP activity that identifies the roles of words or phrases in a sentence relative to the verb, categorizing them into agents, themes, and others based on their semantic features. This approach is useful in various NLP applications, including sentiment analysis, information extraction, and question answering, as it illuminates the connections between different entities and events.

Concisely, our proposed UCDGT tool has demonstrated significant benefits of using the new era AI and machine learning tools in automating various SE processes and achieving the I-CASE principle.

REFERENCES

[1] A. Rasheed, B. Zafar, T. Shehryar, N. A. Aslam, M. Sajid, N. Ali, S. H. Dar and S. Khalid, "Requirement Engineering Challenges in Agile Software Development," *Mathematical Problems in Engineering*, 2021.

[2] M. A. Hagal and F. F. M. Saeid, "A framework for improving the process of discovering potential errors at the requirements engineering stage," in *International Conference on Engineering & MIS*, Istanbul, Turkey, 2022.

[3] J. O. Grady, System requirements analysis, Oxford: Elsevier Inc., 2014.

[4] I. Jacobson, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.

[5] I. Sommerville, Software Engineering, 10th ed., Essex, England: Pearson, 2021.

[6] A. Cockburn, Writing effective use cases, Upper Saddle River, NJ: Addison-Wesley Professional, 2000.

[7] E. M. Jebril, A. T. Imam and M. Al-Fayuomi, "An Algorithmic Approach to Extract Actions and Actors (AAEAA)," in *Proceedings of the International Conference on Geoinformatics and Data Analysis*, Prague, Czech, 2018.

[8] I. K. Raharjana, D. Siahaan and C. Fatichah, "User Stories and Natural Language Processing: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 53811-53826, 2021.

[9] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, H. Peng, J. L. J. W. Z. L. P. Xie, C. Xiong, J. Pei and L. S. Philip S. Yu, "A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT," ArXiv, 2023.

[10] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *The 17th Annual Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, Minnesota, 2019.

[11] H. He, Q. Ning and D. Roth, "QuASE: Question-Answer Driven Sentence Encoding," in *The 58th Annual Meeting of the Association for Computational Linguistics*, Online, 2020.

[12] N. S. Keskar, B. McCann, C. Xiong and R. Socher, "Unifying Question Answering, Text Classification, and Regression via Span Extraction," arXiv, 2019.

[13] A. Radford, K. Narasimhan, T. Salimans and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," Papers with Code, 2018.

[14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, "Language Models are Unsupervised Multitask Learners," Papers with Code, 2019.

[15] J. Li, T. Tang, W. X. Zhao and J.-R. Wen, "Pretrained Language Models for Text Generation: A Survey," arXiv, 2021.

[16] R. Dijkman and S. Joosten, "An Algorithm to Derive Use Case Diagrams from Business Process Models," in *6th IASTED International Conference on Software Engineering and Applications*, Cambridge, United States, 2002.

[17] R. Giachetti, K. Holness and M. McGuire, "The Ability of Engineers to Extract Requirements from Models," in *IEEE 26th International Requirements Engineering Conference (RE)*, Banff, AB, Canada, 2018.

[18] I. S. Bajwa and I. Hyder, "UCD-generator - a LESSA application for use case design," in *International Conference on Information and Emerging Technologies*, Karachi, Pakistan, 2007.

[19] D. K. Deeptimahanti and M. A. Babar, "An Automated Tool for Generating UML Models from Natural Language Requirements," in *IEEE/ACM International Conference on Automated Software Engineering*, Auckland, New Zealand, 2009.

[20] M. Z. Alksasbeh, B. Alqaralleh, B. Alqaralleh, T. A. Alramadin, K. Alemerien and K. Alemerien, "An Automated Use Case Diagrams Generator From Natural Language Requirements," *Journal of Theoretical and Applied Information Technology,* vol. 95, no. 5, pp. 1182-1190, 2017.

[21] N. Arman and S. Jabbarin, "Generating Use Case Models from Arabic User Requirements in a Semiautomated Approach Using a Natural Language Processing Tool," *Journal of Intelligent Systems,* vol. 24, no. 2, p. 277–286, 2014.

[22] A. Alhroob, A. T. Imam and R. Al-Heisa, "The use of artificial neural networks for extracting actions and actors from requirements document," *Information and Software Technology,* vol. 101, pp. 1-15, 2018.

[23] A. T. Imam, A. Alhroob and W. J. Alzyadat, "SVM Machine Learning Classifier to Automate the Extraction of SRS Elements," *International Journal of Advanced Computer Science and Applications,* vol. 12, no. 3, pp. 174-185, 2021.

[24] B. Priyankar, S. Srinivasan, W. C. Sleeman, J. P. IV, R. Kapoor and P. Ghosh, "A Survey on Recent Named Entity Recognition and Relationship Extraction Techniques on Clinical Texts," *Applied Sciences,* vol. 11, no. 18, 2021.

[25] Y. Meng, Y. Zhang, J. Huang, C. Xiong, H. Ji, C. Zhang and J. Han, "Text Classification Using Label Names Only: A Language Model Self-Training Approach," in *The 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online, 2020.

[26] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. R. R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. v. Platen, C. Ma, Y. Jernite, J. Plu, C. Xu and T. L. Scao, "Transformers: State-of-the-Art Natural Language Processing," in *The 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online, 2020.

[27] A. T. Imam, A. J. Al-Nsour and A. Al-Hroob, "The Definition of Intelligent Computer Aided Software Engineering (I-CASE) Tools," *Journal of Information Engineering and Applications,* vol. 5, no. 1, pp. 47-56, 2015.

[28] N. Japkowicz and M. Shah, Evaluating learning algorithms: a classification perspective., NY, USA: Cambridge University Press, 2014.

[29] D. Bajaj, A. Goel, S. C. Gupta and H. Batra, "MUCE: a multilingual use case model extractor using GPT-3," *International Journal of Information Technology,* vol. 14, p. 1543–1554, 2022.

[30] V. S, S. Aithal and P. Desai, “An approach towards automation of requirements analysis,” in *the International MultiConference of Engineers and Computer Scientists*, Hong Kong, 2009.

[31] A. Umber and I. S. Bajwa, “A Step Towards Ambiguity Less Natural Language Software Requirements Specifications,” *International Journal of Web Applications,* vol. 4, pp. 12-21, 2015.