# Advancing Class Diagram Extraction from Requirement Text: A Transformer-Based Approach

**Shweta** and **Suyash Mittal** and **Suryansh Chauhan**
Department of Computer Science and Engineering
The LNM Institute of Information Technology, Jaipur, Rajasthan, India
shweta.singh332@gmail.com

## Abstract

The class diagram plays an important role in software development. As these diagrams are created using software requirement text, it helps to improve communication between the developers and the stakeholders. Thus, the automatic extraction of class diagrams enhances the speed of software development procedures. The research carried out in this direction mostly relies on rule-based methodologies and deep learning models. These methodologies have their drawbacks, such as the fact that large rule-based systems are complex to handle, whereas the word embeddings used in deep learning models are context-independent. Thus, the presented research work strives to extract the class diagram entities from the natural language text by employing a transformer-based model, as the embeddings generated by these models are context-dependent. The results have been compared with the existing procedure, and an ablation study has also been carried out to find out the relevance of each step in the extraction procedure. The analysis involved examining the true positive, false positive, and false negative rates for specific class diagram elements in separate case studies. As a result, an enhancement of 9–7% has been observed in the procedures used for extracting the resulting class diagrams.

## 1 Introduction

The Unified Modeling Language (UML) is a significant aspect of the software requirement specification document. These models are pivotal points for enhancing pertinent communication between software developers and customers. Thus, the modeling ameliorates software requirements and expedites product development according to the requirements of customers. Thus, the automated process of converting software requirement document text into Unified Modeling Language (UML), such as class diagrams, speeds up the adaptation process of requirements according to discussions. The existing literature indicates that different automated methods were proposed for UML model extraction from the requirement text. These processes prominently rely on rule-based methodology (Thakur and Gupta, 2016), (Shweta et al., 2018), and a limited number of them are utilizing neural network architectures (ALH, 2018). However, these existing methods utilize neural networks such as RNN and LSTM, which again take input as word embedding generated through Word2vec (Madala et al., 2020). The word2vec model does not consider contextual information and generates a single-word embedding for each word. Consider an example of the word *'bank'* that can have two different meanings based on context, such as *'riverbank'* or *'bank-a financial institution'*.

However, a transformer-based model will utilise context-based word embedding, and in recent advancements, transformers show remarkable improvement in terms of accuracy and efficient reduction for the computational requirements (Fu and Tantithamthavorn, 2022),(Kalyan et al., 2021). Therefore, it motivates us to assess the effectiveness of a transformer-based language model in extracting class diagrams from software functional requirements. In the proposed approach, we have employed the Robustly Optimized BERT Pretraining Approach (RoBERTa) (Liu et al., 2019) for extraction of class diagram entities. The proposed approach has also been compared with existing approaches to evaluate the performance and effectiveness of the RoBERTa model.

### 1.1 Contribution

The contributions of the proposed work are as follows:

- **Explore the Transformer-based approach:** This research strives to propose an extraction tool with enhanced efficiency of extracting class diagram elements such as classes, attributes, methods, and associations from

textual requirements using RoBERTa-based methodology.

- **Addressing Data set Problems:** We have used a curated data set containing varied software functional requirements, where tokens of each software requirement are labelled with class diagram entities. This research explores multiple aspects to enhance the accuracy of the class diagram entity extraction process, including mitigating biases within the data set.

The rest of the paper is structured as follows: Section 2 presents a concise literature survey of existing extraction procedures. Section 3 explains the proposed methodology step by step taken to reach the current fine-tuned model. Then, section 4 elaborates on our findings, and section 5 explains the significance of each step of the proposed methodology through the ablation study. Finally, the last section 6 contains our concluding remarks along with some future directions.

## 2 Related Works

The literature suggests that existing research has a predominant emphasis on heuristic rules for model extraction from textual requirements. Neural network-based studies in this area are notably scarce. In this section, we present a concise overview, which is divided into two segments. The first segment is dedicated to the extraction procedures focused on heuristic rules, while the second part delves into neural network-based extraction approaches.

The literature suggests that several works have been reported that have utilized extraction rules procedures for the extraction of class diagrams. Researchers employed heuristic extraction rules for UML model entities (Perez-Gonzalez and Kalita, 2002), (Harmain and Gaizauskas, 2003), and some used universal dependencies for the extraction of class diagrams from requirement text (Sagar and Abirami, 2014). In the existing work, the developed extraction rules were formulated based on the grammatical structure of the sentences (Thakur and Gupta, 2016), (Shweta et al., 2018).
In summary, the extraction procedure based on extraction rules has its pros and cons (Gonzalez and Dankel, 1993), (McGarry et al., 1999). Rules can be easily removed or added to a rule-based system, and it does not require a large amount of corpus. However, a rule-based system with a large number of rules is difficult to maintain (Gonzalez and Dankel, 1993). Thus, some researchers also employed neural network architecture for the extraction of different models from requirement text, such as authors identifying component state transition diagrams from requirements using recurrent neural network (RNN) and long short-term memory (LSTM) architecture. Similar work is also carried out to automatically identify actors and actions in a system's requirements using neural networks (ALH, 2018). In (Pudlitz et al., 2019), a self-trained named-entity recognition model is used with bidirectional LSTMs and CNNs to extract the system states from requirements text. In (Madala et al., 2020), authors found that the neural network RNN with LSTM and RNN with GRU architectures provide good performance in the identification of model elements, but the results changed in different conditions. Hence, we can conclude that the transformer-based approach has not been utilized and tested until now for the model extraction process from the requirement text.

## 3 Proposed Methodology

The proposed methodology can be divided into mainly two components, i.e., data preprocessing and training of the model, as shown in Fig. 1.

### 3.1 Dataset Annotation-Tagged XML Dataset

The first step of our proposed methodology involves the annotation of the data set. We considered the PURE data set (Ferrari et al., 2017) for our experiment. We have utilized a data set that includes thirty-two requirement documents and 34,268 sentences. The functional requirements are separated from SRS documents and then passed to volunteers to annotate the class diagram entities within the functional requirements. The volunteers included graduate students, post-graduate students, Ph.D. candidates, and some industry experts. Students employed GATE (General Architecture for Text Engineering) (Cunningham et al., 2013) to annotate the requirements that generate the tagged XML data sets. The annotation carried out by students has been finalized as per the reviews provided by the experts.

The rest of this section provides a detailed description of these two components in a step-by-step manner, thereby providing a comprehensive understanding of our approach to automating the software design process through LLMs.

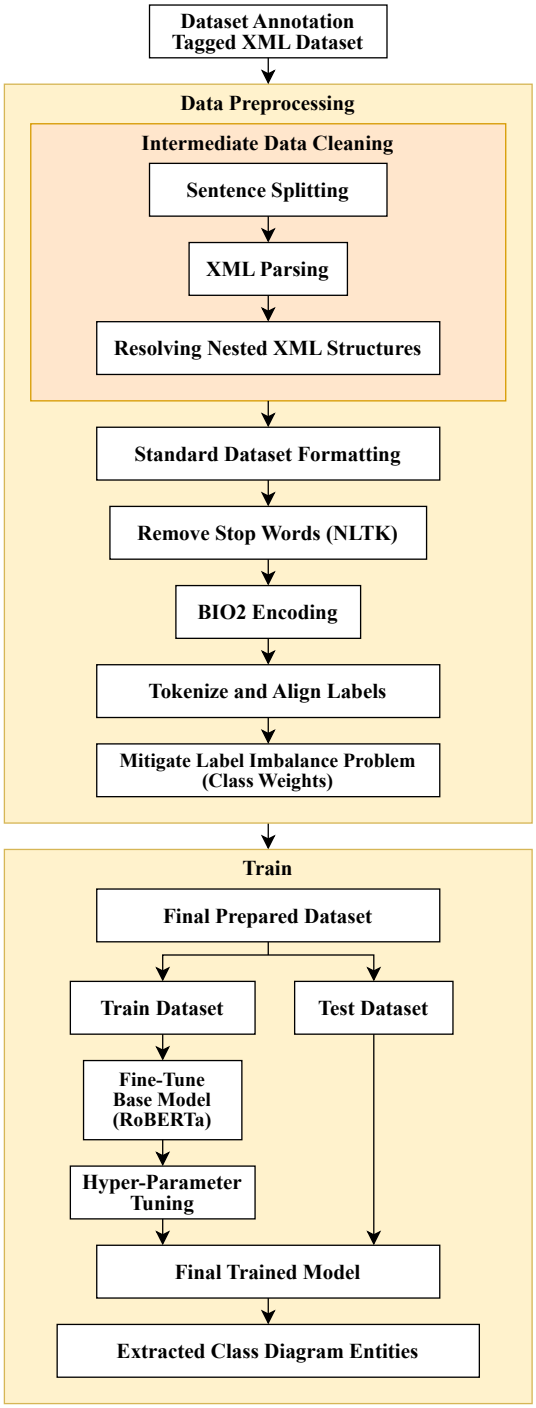


Figure 1: Proposed Methodology

## 3.2 Data Preprocessing

The first step of the preprocessing process includes intermediate data cleaning. This step contains sentence splitting, XML parsing, and fixing structure resolution errors. These steps are explained as follows:

### 3.2.1 Intermediate Data Cleaning

Data cleaning is a critical step to extract pertinent textual information from the problem statements and prepare them for subsequent processing. We started with an XML-based tagged data set, on which we performed the following intermediate steps:

1. **Sentence Splitting**: Since the XML data is in a hierarchical structure, each specification was divided into sentences to preserve the contextual information hinting to the LLM about their respective meanings and relationships.

2. **XML Parsing**: This step involved extracting and organizing data from the XML hierarchical structure into a sequenced data set format. XML parsing makes the information more accessible and structured. It facilitates further analysis and understanding while preserving the context of the problem statement and helping in batch training.

3. **Resolving Nested XML Structures**: This step involves handling any erroneous XML tags that caused the nesting of labels. This is done to ensure that the data aligns with the specific requirements of the class diagram entity task.

### 3.2.2 Standard Dataset Formatting

In this step, the dataset format is changed into a standard format that can be easily fed to the LLM as input. This is carried out to ensure uniformity and compatibility with the extraction of class diagram entities.

### 3.2.3 Remove Stop Words

Stop words (Sarica and Luo, 2021) usually include articles, prepositions, conjunctions, and pronouns that have no or very little significance in the actual meaning of the sentence. A list of such stop words was obtained from the (Bird et al., 2009) library. All occurrences of these words were stripped of their current class. This removes the unnecessary words, helping the language model focus on actual data.

### 3.2.4 Inside-outside-beginning (IOB)2 Encoding

IOB2 encoding is used to tag the tokens for chunking tasks, such as token classification tasks similar to our class diagram entity identification. The IOB2 scheme is widely employed to represent entity spans with precision to enhance the model's learning capabilities. This step involved assigning I, O,

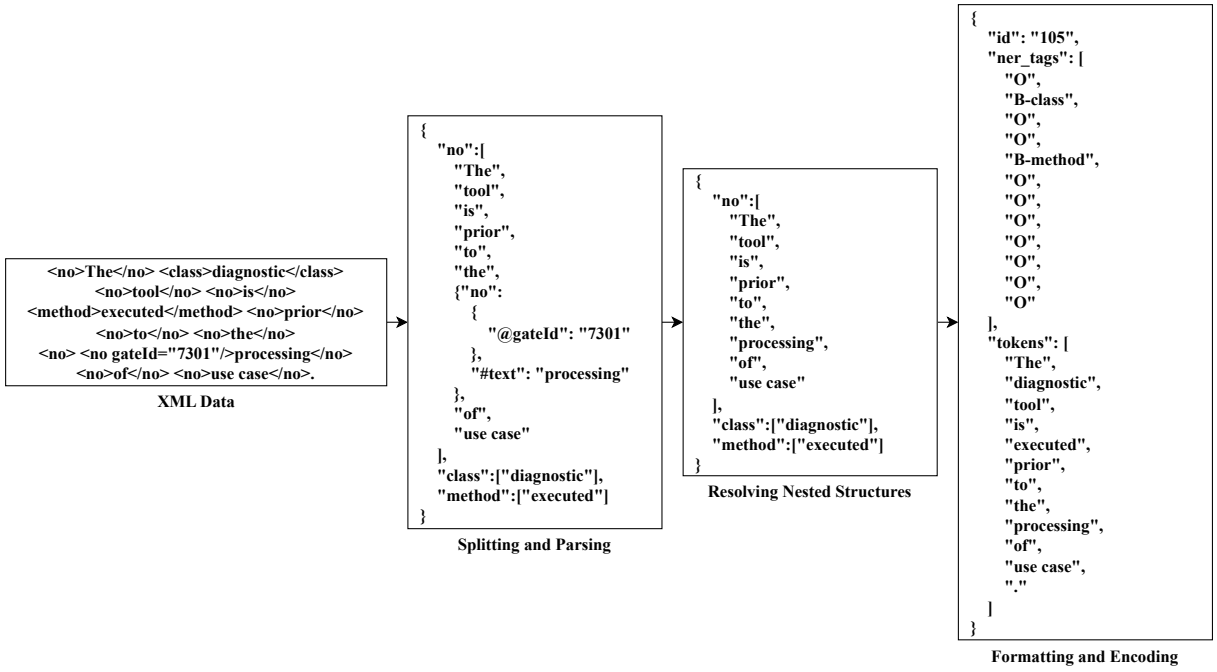


Figure 2: Steps of Preprocessing shown using an Example

and B labels for these six labels: *'Others', 'class', 'attribute','method', 'association', and 'generalization'*.

- **B**eginning: Any token belonging to a different known entity than the previous token is tagged as *B-label*.
- **I**nside: Any tokens consecutive to the token tagged as B and of the same known entity type are tagged as *I-label*.
- **O**utside: Those tokens that do not belong to any known entity are tagged as *Other*.

The preprocessing steps, including intermediate data cleaning, formatting, and cleaning, are illustrated in Fig. 2 using an example.

#### 3.2.5 Tokenize and Align Labels

This step uses the *AutoTokenizer* of the RoBERTa model to tokenize the inputs. Sometimes, there is the possibility of multiple splittings of words. In this scenario, simultaneous alignment is necessary for these tokens to have their labels. Thus, Algorithm 1 is employed to resolve this issue. The algorithm labels only the first token of the word and assigns a negative value to other special tokens and the remaining tokens of the word. This is an indication for the model to ignore these negatively valued tokens in entity analysis and specifically classify beginning tokens only.

#### 3.2.6 Mitigate Label Imbalance Problem

Weights are assigned to each label corresponding to their occurrence frequency to address the issue of unbalanced label distribution. This is done to ensure that: 1) Labels with lower frequency counts are not underrepresented and are given equal importance while training. 2) The model will no longer be biased towards labels with high frequency when tagging unseen data.

### 3.3 Training

#### 3.3.1 Train-Test Split

After preprocessing, the obtained dataset is split into training (85%) and testing (15%) data sets to avoid overfitting.

#### 3.3.2 Fine Tune Model

The RoBERTa Large Language Model was fine-tuned on the training dataset by adding a new layer on top of the existing architecture, enabling it to predict labels for tokens in the input sequence. Due to space constraints, we are not able to add a description of the RoBERTa model to this manuscript.

The training process occurred over multiple epochs, where the data was fed in batch sizes of 8. This is quite low by general standards, but it was mandated by the current GPU training capabilities.

**Algorithm 1:** Tokenize and Align Labels

**input :** record: a dataset record containing tokens and their corresponding tags

**output :** tokenized_inputs: a dataset record of tokenized inputs with their aligned labels

**begin**

$tokenized\_inputs \leftarrow tokenizer(R[tokens])$

$labels \leftarrow \emptyset$

**for** $labels \in R[\text{"}ner\_tags\text{"}]$ **do**

$word\_ids \leftarrow tokenized\_inputs.word\_ids()$

$previous\_word\_idx \leftarrow \emptyset$

$label\_ids \leftarrow \emptyset$

**for** $word\_idx \in words\_ids$ **do**

**if** $word\_idx$ *is* $\emptyset$ **then**

$labels\_ids.Append(-100)$

**end**

**else if** $word\_idx \neq previous\_word\_idx$ **then**

$label\_ids.Append(label[word\_idx])$

**end**

**else**

$labels\_ids.Append(-100)$

**end**

$previous\_word\_idx \leftarrow word$

**end**

**end**

$labels.Append(label\_ids)$

$tokenized\_inputs \leftarrow labels$

**return** $tokenized\_inputs$

**end**

#### 3.3.3 Hyper-parameter tuning

Hyper-parameter tuning was executed to find the best values for training parameters like learning rate, weight decay and max-steps. Different optimizers (for optimizing the weights and biases) like *adam_torch*, *adam_hf*, and *adam_torch_fused* were also experimented with. Eventually, we settled upon *adam_torch* (Kingma and Ba, 2017), as it was the best performer.

#### 3.3.4 Extract Entities from Test Dataset

The trained model has been saved and is hence ready to extract entities from the testing dataset. Then, each sentence was passed into the model, and the predicted labels were recorded to evaluate the trained model using metrics.

The dataset and final trained model are saved for running inferences, downloading, or deploying as APIs. We are not able to provide that tool link to remain anonymous; however, on request, we can provide that also.

## 4 Results and Discussions

The aim of this section is to evaluate the proposed model using metrics such as precision, recall, and accuracy. Each of these evaluation metrics is explained as follows:

### 4.1 Evaluation Metrics

**Precision**: measures the amount of false positives. It lies between 0 and 1, 1 signifying no false positives.

$$P = TP/(TP + FP)$$

**Recall**: measures the amount of false negatives. It lies between 0 and 1, 1 signifying no false negatives.

$$R = TP/(TP + FN)$$

**F1-Score**: is the harmonic mean of precision and recall:

$$F1 - Score = 2 * P * R/(P + R)$$

### 4.2 Comparison With Existing Rule-Based Methodologies

We employed two case studies to illustrate the distinction between the existing extraction procedure and the newly proposed methodology. The existing methodology (Shweta et al., 2021) is a recent extraction work based on a rule-based approach that is able to extract all the class diagram elements from software requirements. Thus, it makes this methodology suitable for comparison with the proposed approach. As depicted in Table 1, the transformer-based methodology has the upper hand with respect to rule-based methodology. The accuracy of class extraction has increased by an average of 7%, and the accuracy of attribute extraction has improved by an average of 9%. However, there is only a slight enhancement in the extraction of methods and associations, with average improvements of 5% and 1.5%, respectively.

The extraction of classes is more precise in comparison to other class diagram elements, including methods, associations, and attributes. The extraction results of the proposed methodology for the library management system are shown in Figure 3.

Table 1: Comparison of Methodologies

| Models | Metrics | Library Management System (Ahsan Riaz) | | | | Airport System (S.S.S., 2012) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | *C* | *AT* | *M* | *AS* | *C* | *AT* | *M* | *AS* |
| **RoBERTa** | *TP* | 7 | 8 | 22 | 4 | 8 | 6 | 0 | 5 |
| | *FP* | 0 | 0 | 6 | 1 | 0 | 1 | 0 | 1 |
| | *FN* | 0 | 4 | 1 | 0 | 0 | 0 | 0 | 2 |
| | *Precision* | 1.00 | 1.00 | 0.78 | 1.00 | 1.00 | 0.86 | 1.00 | 0.83 |
| | *Recall* | 1.00 | 0.67 | 0..95 | 0.80 | 1.00 | 1.00 | 1.00 | 0.71 |
| | *F1-Socre* | 1.00 | 0.80 | 0.86 | 0.88 | 1.00 | 0.93 | 1.00 | 0.76 |
| **Intermedi-ate Template Based Approach (Shweta et al., 2021)** | *TP* | 6 | 4 | 21 | 4 | 7 | 6 | 0 | 4 |
| | *FP* | 1 | 0 | 9 | 1 | 1 | 2 | 0 | 1 |
| | *FN* | 0 | 4 | 4 | 0 | 1 | 0 | 0 | 2 |
| | *Precision* | 0.85 | 1.00 | 0.70 | 1.00 | 0.88 | 0.75 | 1.00 | 0.80 |
| | *Recall* | 1.00 | 0.50 | 0.84 | 0.80 | 0.88 | 1.00 | 1.00 | 0.67 |
| | *F1-Score* | 0.91 | 0.67 | 0.76 | 0.88 | 0.88 | 0.86 | 1.00 | 0.73 |

C - Class, AT - Attribute, M - Method, AS - Association

This shows the requirements of the library management system along with the extracted class diagram entities using the proposed model.

The requirements text suggests that if 'LMS' is subject to every statement of requirements, then 'LMS' should be extracted as a class according to the rule, i.e., 'The subject should be extracted as a class.'. If 'LMS' is extracted as a class for the class diagram, then all the extracted methods will be associated with this single class only, and ultimately, it will increase the complexity of the system. Thus, in good practice, LMS should not be extracted as a class. In this scenario, again, we have to write some other rules to avoid this problem. Thus, it will again increase the complexity of a rule-based system. The same problem is resolved by using the proposed transformer-based approach by training the model on this type of statement, and the results show great improvement in such kinds of problems of requirement documents.

It is evident from Figure 3 that 'LMS' is not extracted as a class.

## 5 Ablation Study

The ablation study has been performed to find out the importance of every step of the proposed methodology. The following subsections demonstrate the significance of each part of the methodology.

### 5.1 Initial Step

We started with the implementation of DistilBERT (uncased) (Sanh et al., 2020) as the base model. DistilBERT's efficiency and competitive performance made it a suitable starting point from which we could try numerous techniques. A decision was made to use the cased version, as there would be an impact on the class extraction between capitalized and non-capitalized words. The DistilBERT model tries to classify every token into a label. For example, the word 'login' was split into 'log' and 'in', and each part was identified as belonging to a different class, both of those classes being incorrect. DistilBERT performed fairly well (76% Accuracy) given the smaller dataset that it was originally trained on. However, other metrics like precision and recall are quite low, suggesting that it makes more false classifications.

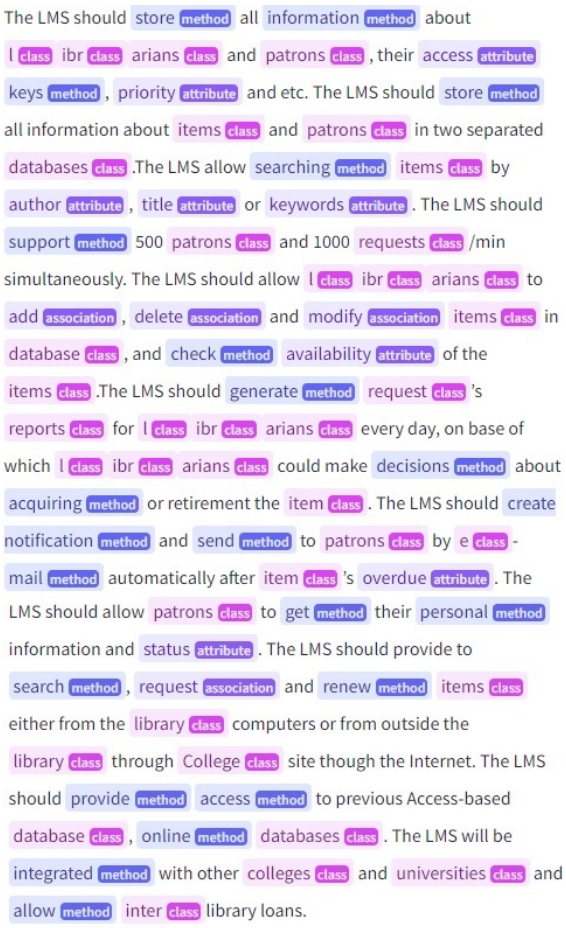


Figure 3: Sanpshot of extracted classes diagram elements in Library Management System

### 5.2 Model Change to BERT

We transitioned from DistilBERT to BERT (Devlin et al., 2018) (with a larger parameter count and model size) to enhance the accuracy and precision of entity recognition within our data. The decision to switch to BERT was based on the understanding that BERT's deep learning architecture, which considers both the left and right context of words in a sentence, could significantly improve our entity recognition results. By capturing these complex contextual nuances, BERT can more accurately identify and classify entities within our data, ultimately leading to more precise and context-aware outcomes.

The BERT model performed slightly worse (74% accuracy) than DistilBERT, but that is most likely due to the small batch size during training due to limited GPU training resources.

### 5.3 Mitigation of Label Bias Problem

Recognizing the imbalance in the distribution of entity labels within the dataset, we introduced weighted labels during training. This measure ensured that the language model assigned appropriate attention to underrepresented entities.

Doing so improved the accuracy by almost 5%. A substantial increase in the F1 Score (60%) can be seen, implying that the model is now making fewer false classifications.

### 5.4 Removal of Stop Words

On analysis of the dataset, one could see words that were not pertinent to entity recognition. To enhance accuracy, we implemented a token set transformation, removing these stop words. This helped remove irrelevant jargon from the dataset, which masked the effect of tokens containing relational data of the functional requirement. The effect of the removal of stopwords on the label distribution can be visualized in Fig. 4.

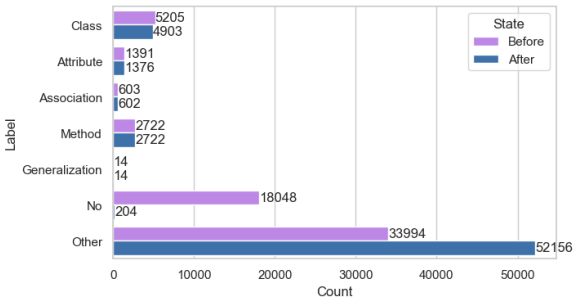


Figure 4: Effect of Removing Stop Words on Label Distribution

This made the most significant impact, increasing the accuracy to 86% and having the highest Recall of 74%. Loss during training was also appreciably low.

### 5.5 Model Change to RoBERTa

In pursuit of superior results, we continued to augment the model's complexity. We transitioned to RoBERTa, an advanced variant of BERT, which provided an enhanced understanding of contextual nuances (Liu et al., 2019).

The choice to transition to RoBERTa was because of its substantially larger parameter and model size compared to BERT. This expanded size equips our model with a greater capacity to capture intricate linguistic nuances.

After making this switch, accuracy peaked at 89% with the best overall precision, recall, and F1-Score, as shown in Table 2. The results of the ablation study can also be visualized using Fig. 5.

| Model | Precision | Recall | Accuracy |
|---|---|---|---|
| DistilBERT | 0.51 | 0.46 | 0.76 |
| BERT | 0.49 | 0.53 | 0.75 |
| BERT - W | 0.54 | 0.67 | 0.79 |
| BERT - SW | 0.44 | 0.75 | 0.86 |
| RoBERTa | 0.60 | 0.75 | 0.89 |

W - Weights, SW - Stop Words

Table 2: Model Wise Metrics

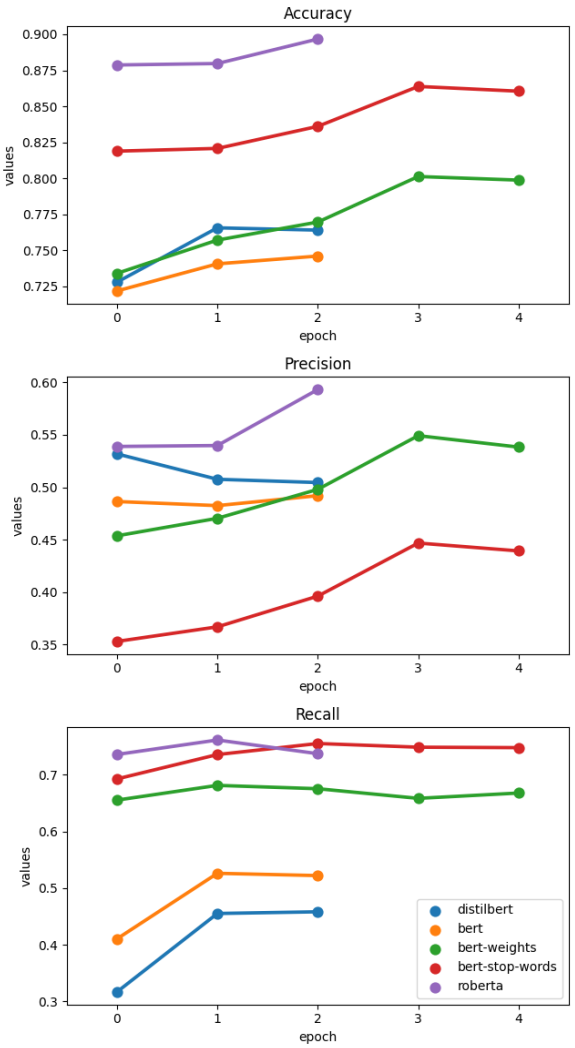


Figure 5: Training Metrics

While the proposed method has demonstrated promising results, there is room for further improvement. This can be achieved through several avenues: 1) **Expanding the Dataset:** As of now, only the PURE dataset has been used for fine-tuning the model. To enhance the model's performance, the dataset can be expanded to encompass a broader spectrum of software engineering domains and complex problem statements. 2) **Enhanced Computational Resources:** Acquiring additional computational resources can help approach the upper limits of performance. Fine-tuning a base model with more parameters and a larger scope can contribute to improved results. 3) **Advanced Techniques:** Implement techniques such as cross-validation and extensive hyperparameter tuning to refine the current results and optimize the model's performance. 4) **Post-processing Refinement:** The encoding used in RoBERTa leads to some incorrect tagging, i.e., 'cart' and 'item' are extracted as two different classes, so to make it correct, these two words should be merged. Thus, further refinement in the post-processing of the extracted class diagram entities is necessary to ensure the accuracy and consistency of the results. 5) **Complete Class Diagrams:** Improving accuracy can also be accomplished by constructing complete class diagrams that connect all the extracted elements, i.e., to correctly identify the connection between the classes and their attributes, methods, and associations with other classes. This will result in a clearer and more comprehensive representation.

These strategies can collectively lead to a more robust and effective methodology for class diagram extraction and analysis.

## 6 Conclusion

In the current study, we proposed a new approach to automate the software design process through language model-based entity analysis for class diagram formation in an attempt to address the limitations of the existing approaches. Transformer-based models are also able to consider the context of the words while extracting the class diagram elements and, hence, achieve better results for previously unseen statements. The tool, although quite effective, remains a tool and must be human-assisted to maintain integrity and avoid edge cases that may hamper the planning process. There may exist cases that hold very correlated information that may be hard to disassemble correctly by the transformer-based model. This remains a limitation of the approach, along with its extensive dependence on the trained dataset, with inaccuracies being noticeable in software specifications not previously seen by the model. In conclusion, our research presents a promising approach to automating the software design process, significantly reducing the cognitive load on software architects while improving the quality of software designs.

## References


2018. The use of artificial neural networks for extracting actions and actors from requirements document. *Information and Software Technology*, 101:1–15.

Hamaad Chuadry Ahsan Riaz, M.Bilal sp13-bse-106. Library management system.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Hamish Cunningham, Valentin Tablan, Angus Roberts, and Kalina Bontcheva. 2013. Getting more out of biomedical documents with gate's full lifecycle open source text analytics. *PLoS Comput Biol*, 9(2):e1002854.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. 2017. Pure: A dataset of public requirements documents. In *Requirements Engineering Conference (RE), 2017 IEEE 25th International*, pages 502–505, Lisbon, Portugal. IEEE.

Michael Fu and Chakkrit Tantithamthavorn. 2022. Gpt2sp: A transformer-based agile story point estimation approach. *IEEE Transactions on Software Engineering*, 49(2):611–625.

Avelino J. Gonzalez and Douglas D. Dankel. 1993. *The Engineering of Knowledge-Based Systems: Theory and Practice*. Prentice-Hall, Inc., USA.

H.M. Harmain and R. Gaizauskas. 2003. Cm-builder: A natural language-based case tool for object-oriented analysis. *Automated Software Engineering*, 10(2):157–181.

Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. 2021. Ammus: A survey of transformer-based pretrained models in natural language processing. *arXiv preprint arXiv:2108.05542*.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Kaushik Madala, Shraddha Piparia, Eduardo Blanco, Hyunsook Do, and Renee Bryce. 2020. Model elements identification using neural networks: a comprehensive study. *Requirements Eng.*

Kenneth McGarry, Stefan Wermter, and John MacIntyre. 1999. Hybrid neural systems: from simple coupling to fully integrated neural networks. *Neural Computing Surveys*, 2(1):62–93.

Hector G. Perez-Gonzalez and Jugal K. Kalita. 2002. Gooal: A graphic object oriented analysis laboratory. In *Companion of the 17th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 38–39), New York, NY, USA. ACM.

Florian Pudlitz, Florian Brokhausen, and Andreas Vogelsang. 2019. Extraction of system states from natural language requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 211–222.

Vidhu Bhala R. Vidya Sagar and S. Abirami. 2014. Conceptual modeling of natural language functional requirements. *Journal of Systems and Software*, 88:25 – 41.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Serhad Sarica and Jianxi Luo. 2021. Stopwords in technical language processing. *PLOS ONE*, 16(8):e0254937.

Shweta, R. Sanyal, and B. Ghoshal. 2018. Automatic extraction of structural model from semi structured software requirement specification. In *Proc. IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, pages 543–58), Singapore.

Shweta, Ratna Sanyal, and Bibhas Ghoshal. 2021. Automated class diagram elicitation using intermediate use case template. *IET Software*, 15(1):25–42.

S.S.S. 2012. Class diagram for airport uml questions.

Jitendra Singh Thakur and Atul Gupta. 2016. Anmodeler: A tool for generating domain models from textual specifications. In *Proc. 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 828–833), New York, NY, USA. Association for Computing Machinery.