A Transformer Based Pipeline for Software Requirements Classification

1st Maheen Mashrur Hoque
Department of Computer Science and Engineering
Islamic University of Technology
Gazipur, Bangladesh
email address or ORCID

2nd Arman Hossain Dipu Khan
Department of Computer Science and Engineering
Islamic University of Technology
Gazipur, Bangladesh
email address or ORCID

3rd Ahsan Habib
Department of Computer Science and Engineering
Islamic University of Technology
Gazipur, Bangladesh
email address or ORCID

4th Ajwad Abrar
Department of Computer Science and Engineering
Islamic University of Technology
Gazipur, Bangladesh
email address or ORCID

***Abstract*—The automation of the software development lifecycle (SDLC) is a key challenge in the field of software development. One of the main challenges in automation of the SDLC lies within the work of collecting, analyzing and establishing requirements, where the requirements collected from the stakeholders are often noisy, which can be difficult to organize. In order to facilitate the automation of requirement analysis, our study proposes two cost effective approaches, the first being leveraging the lightweight DistilBERT and RoBERTA models with a method called "Ensemble Pooling" to filter out relevant requirements, and the second one being a retrieval augmented generation (RAG) based classification leveraging GPT-4 and ChromaDB vector store to discriminate between functional and non-functional requirements. The experiments conducted by us showcased an accuracy of 78% for the first approach and 86.67% in the second approach.**

***Index Terms*—Software Requirements, Software Development Lifecycle (SDLC), Classification, Natural Language Processing (NLP), Transformers, Retrieval Augmented Generation (RAG), Ensemble**

I. Introduction

Software requirements can be thought of very high level description of what a software system is expected to do in order to solve a business requirements and are the reflection of the expectations of the stakeholders and clients [**?**]. Software requirements are typically divided into three categories [**?**] – Functional (Specify the purpose of the system), Non-Functional (Define the system's quality attributes such as performance), and Domain Requirements (Specific to the context of the application domain such as memory requirement). A requirement should have clarity, be consistent and have completeness [**?**]. However, factors like communication gap, difference in perspective, and complexity of scope can make requirements vague and difficult to comprehend for the developer [**?**]. Due to these factors, requirements from the stakeholders may often contain irrelevant information (which we are referring to as "noise"), and it can be difficult to discern functional and non-functional requirements. This causes bottleneck in the SDLC as system architects spend much time organizing the requirements. The challenge of requirements organization can be classified as a natural language processing (NLP) problem. Hence, our research objective (RO) in this work is to deal with two critical steps:

- **RO1:** Noise reduction via classifying relevant and irrelevant requirements.
- **RO2:** Improving requirement assessment via functional and non-Functional requirements.

In both objectives, we experimented with two different approach for classification, with the works of Ivanov et. al. [**?**] acting as baseline. The experiments, dataset and results are discussed in later sections.

II. Literature Review

This section provides a comprehensive descriptions of the previous works that were reviewed for this research. In order to avoid parochial view over the research objective, the reviews were not limited to only language model based approaches.

A. Esoteric Approach

The works of Siahaan and Darnoto distinguishes irrelevant requirements using the MultiPhiLDA method [**?**]. Their method works by distinguishing topic–word distribution of actor words and action words, governed by polynomial probability functions, essentially making it a statistical distribution analysis.

Another method by Alharbi et. al. leverages semantic embeddings to filter ambiguous requirements [**?**]. The embedding were fine-tuned to the specific task, allowing it to capture context-specific nuances related to ambiguity.

Similarly, the works pf Malik et. al. leverages cosine similarity over sentence embeddings to classify conflicting requirements [**?**]. Although, not directly related, this work provided us insight on how to levarage similarity scores.

B. Language Model Approach

A language model based approach by Ivanov et. al. for extracting requirements from unstructured text uses BERT [**?**] model fine-tuned with a manually annotated dataset from the PURE (Public Requirements) corpus [**?**]. They achieved an accuracy of 74%, compared to two other baseline approaches – fastText (open source natural language processing toolkit) with an accuracy of 60%, and ELMo (a foundational text-embedding model) paired with SVM (Support Vector Machine classifier) with an accuracy of 59%. Due to the similarity of objective, this work was considered as a baseline for us to compare against.

III. Dataset

A. PURE Dataset

For our work the, the PURE (Public Requirements) dataset [**?**], created by Ferrari et. al., was the primary source of data for the experiments of RO1. This contains 79 software requirements document (SRS) that contains 34,286 sentences. However, the dataset does not provide any additional labelling for the requirements sentences to aid natural language processing tasks. Some of the previous works, notably [**?**] and [**?**] utilized this dataset in their works via labeling in various ways.

A labeled subset of the PURE dataset, called the Reg-ExpPURE Dataset by Ivanov et. al. [**?**] was used in our studies. This dataset has proper balancing in it's data, with 2474 not requirements and 2832 requirements in train, 467 not requirements and 1058 requirements in test and 255 not requirements and 650 requirements in validation set, making it a suitable training candidate.

B. FR-NFR Dataset

For the experiments of RO2, a custom dataset was made with SRS documents from three real projects conducted by the Ministry of ICT, The People's Republic of Bangladesh - A Training Database, an Enterprise Management System, and an AI Chatbot. The requirements were collected and labeled into FR (Functional Requirements) and NFR (Non-Functional Requirements) by three expert personnel from the software industry of Bangladesh - two experienced product managers and an experienced AI engineer. To handle labeling conflict, we revisited standard definitions of FR and NFR [**?**], and industry-recognized standards such as IEEE guidelines [**?**] and had discussions until agreement was reached.

IV. Methodology

A. Leveraging the Power of Transformer Models

B. Noise Reduction in Requirements

C. GPT-4 Text Embedding

Our experimentation utilizes a two-step Search-Ask method to enable GPT to answer questions using a library of reference texts. We selected an SRS as an ideal SRS completely complying with IEEE Std 830 (1998) , also used in the paper **PURE: a Dataset of Public Requirements Documents** [**?**] for benchmarking ideal functional and non-functional requirements.

We began by dividing the Software Requirements Specification (SRS) into short, self-contained sections. We generated embeddings for each section using the GPT-4 embeddings API and stored them in a vector database (ChromaDB). ChromaDB can perform efficient semantic search across the entire SRS, even when user queries did not share direct lexical overlap with the document text. To retrieve relevant information, we employed Maximum Marginal Relevance (MMR), which allowed us to balance relevance and diversity in the retrieved segments and avoid redundant or overly similar results.

Once the candidate segments were retrieved, we ranked them according to their semantic similarity to a model-generated hypothetical ideal answer. This ranking process ensured that we selected the most contextually meaningful information for the task. We then inserted the highest-ranked SRS sections directly into the prompt.

With this curated context, we asked the model to interpret the SRS content, resolve ambiguities, and extract both functional and non-functional requirements. By separating the search and generation steps, we avoided the shortcomings of fine-tuning and message-insertion memory techniques, which can introduce factual drift or reduce reliability. Our approach ensured consistent recall of the original SRS while supporting accurate and dependable requirements classification, even when the source material was unstructured or ambiguous.