International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Context Aware Semantic Contract Document Clustering and Labeling System

K. Sushma (227Z1A6670), Firdous (227Z1A6627)
K.L. Abhiram (227Z1A6667)
Bachelor Of Technology N
Computer Science And Engineering (Artificial Intelligence And Machine Learning)
Under the Guidance of

Mr. M. Eswara Rao
Assistant Professor
School Of Engineering
Department Of Computer Science And Engineering
(Aritifical Intelligence And Machine Learning)
Nalla Narasimha Reddy
Education Society's Group Of Institution
(An Autonomous Institution)

ABSTRACT - Contracts are vital legal and business documents, yet managing and analysing them is challenging due to their complex structure, varied formats, and domain-specific terminology. Traditional keyword-based clustering often fails to capture the true meaning of legal texts, leading to poor classification and inefficient management.
This project proposes a Semantic-Based Contract to Document Clustering System that employs Natural Language Processing (NLP) and Machine Learning (ML) techniques to group contracts according to semantic similarity.
The system not only clusters documents with similar clauses and intent but also assigns meaningful names to each cluster by analysing frequent terms and clause patterns, ensuring interpretability and ease of use. By automatically categorizing contracts such as employment agreements, leases, and non-disclosure agreements, the system reduces manual effort, enhances document retrieval, and supports compliance and risk management. Scalable and adaptable, it offers an effective solution for organizations handling large volumes of legal documents.

Keywords: ***Natural Language Processing (NLP), Machine Learning (ML), Document Clustering, Contract Management, Semantic Analysis, Legal Document Processing, Information Retrieval***

LIST OF ABBREVIATIONS

S. NO.	ABBREVIATION	DEFINITIONS
1	NLP	Natural Language Processing
2	ML	Machine Learning
3	AI	Artificial Intelligence
4	TF-IDF	Term Frequency–Inverse Document Frequency
5	BERT	Bidirectional Encoder Representations
		from Transformers
6	API	Application Programming Interface
7	UI	User Interface
8	DB	Database
9	K-Means	K- Means Clustering Algorithm
10	NDA	Non-Disclosure Agreement

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

1. INTRODUCTION

In the current digital environment, organizations have a large number of legal and business documents, among which contracts are of significant importance. These contracts are usually of complex data structure, vary in data format, and contain domain-specific legal terms, making it a challenging task to organize and analyze these contracts. The current contract management system mainly depends on manual processing or keyword-based techniques, which do not take into consideration the context of the data. This has made the classification of contracts inefficient, has made it difficult to retrieve relevant contracts, and has increased the possibility of misinterpretation of contract data.

Considering these challenges, this project has introduced a new concept of a Semantic-Based Contract Document Clustering System that uses advanced techniques of Natural Language Processing (NLP) and Machine Learning (ML). Instead of depending on traditional techniques that do not take into consideration the context of the data, this system has focused on understanding the semantic meaning of contract documents by converting the data into vector embeddings. The system not only groups similar documents but also improves interpretability by automatically assigning meaningful labels to these groups of documents. This is accomplished by analyzing common terms and clause structures found in these groups of documents. Therefore, users can readily identify and interpret the nature of these grouped documents without having to individually examine these documents.

The system not only groups similar documents but also improves interpretability by automatically assigning relevant labels to the groups of documents. This is achieved by analyzing the most common terms and clause structures within each group. This way, one can easily understand the nature of the grouped documents without having to go through each document. Moreover, the proposed system greatly reduces manual effort and improves efficiency in managing large numbers of contracts. The proposed system can be used to efficiently perform document retrieval, organization, and can even aid in compliance and risk management. The proposed system design ensures that it can be used to efficiently handle large numbers of documents and different types of legal documents.

1.1 Motivation

Many of these contracts and legal documents in a modern organization are in a digital format, and thus, it becomes difficult to manage and handle since each one is unstructured, and the language used is difficult to understand and it is too complicated to handle and has terminologies that are specific to the domain. It is time consuming and has errors in processing such documents manually. Conventional methods and techniques, such as TF-IDF, are based on keywords, which miss contextual and semantic meaning of legal text, resulting in inaccuracy in clustering and ineffective retrieval. Transformer-based models like BERT can generate contextual embeddings, which are the actual meaning of text, with the development of Natural Language Processing (NLP). These embeddings enhance the interpretation of the contract clauses and their meaning. This project is motivated by the need to create a system of semantic-based clustering that employs BERT embeddings and cosine similarity to cluster contracts by meaning, as opposed to key words. This will improve the accuracy of clustering, lessen the manual workload and offer a more effective means of managing contracts, retrieving them and analyzing risks at a larger scale efficiently.

1.2 Problem Statement

Modern organizations have high volumes of contracts which are not organized and have complicated legal terminology. It is hard, time-consuming, and cannot be scaled to manage and organize these documents manually. The current systems employ mostly costly methods or techniques such as TF-IDF which are not suitable in capturing the real meaning and context of the text using keywords. Consequently, like contracts might not be clustered at the right location and therefore facilitating poor document retrieval and analysis. Such deficiency of appropriate semantic comprehension complicates finding similar documents and facilitating activities like compliance control and decision-making. Thus, it is necessary to have an automated method that can comprehend the text of contracts and group them according to semantic similarity using sophisticated NLP methods such as BERT embeddings.

1.3 Purpose

This project will be aimed at creating an automated system that will help sort and analyse large amounts of contract documents with the help of state-of-the-art Natural Language Processing (NLP) methods. It is based on the BERT based embeddings to capture the semantic sense of the contracts and cosine similarity to cluster similar documents. The project will set out to cluster the contracts on the basis of the actual circumstances and purpose of the contract, instead of keyword matching. It is also concerned with creating significant labels to each cluster to elevate interpretability and usability. With such a system in place, the amount of manual work is minimized and accuracy of document classification is improved and the effectiveness of the contract

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

retrieval and management is also increased. It also facilitates enhanced compliance, risk evaluation and decision-making within organisations with big legal data.

1.4 Scope

This project covers the development of a contract processing and analysis system that will be based on Natural Language Processing methods and unstructured contract documents. It aims at producing semantic representations of contracts through BERT-based embeddings. Similar documents are grouped by using contextual meaning similarity (cosine similarity) to find similar documents by the system. It embodies preprocessing the process which comprises of text cleaning and normalization. Another aspect of the project is grouping of contracts and coming up with meaningful groups. It also produces labels to each cluster so that they can be interpreted better. The system only analyses in text and does not carry out legal validation. It is predetermined to be scalable, and can be extended to the real world.

1.5 Objective

This project aims at devising a semantic-based contract document clustering system based on Natural Language Processing. It tries to form contextual embeddings with the help of BERT to encode the meaning of contracts and use cosine similarity to group them accurately. Another area that is covered in the project is on how to enhance document organization and retrieval. Moreover, it aims at eliminating manual work and the ability to scale up to handle great numbers of legal documents. It also strives to produce meaningful labels of each cluster in order to increase the interpretability. Besides, the system facilitates improved compliance and decision-making analysis.

1.6 Limitations

The system is dependent on the quality and availability of the input data, and thus inaccurate or incomplete documents can influence the clustering accuracy. Only text is analyzed but other non-text features like images or signatures are ignored. In some cases, the model might not be able to embrace extremely sophisticated or ambiguous legal language. Pre-trained BERT embeddings are utilized in the project, and they might not be optimized to all legal areas without additional fine-tuning. Neither does it conduct a legal validation or check the appropriateness of the content of contracts. Moreover, large datasets may need fairly high levels of computational resources to embed generation.

2. LITERATURE SURVEY

2.1 Introduction

Previous research in document clustering and text analysis has mainly concentrated on keyword-based and statistical approaches. Methods like TF-IDF and Vector Space Modeling have been extensively used for numerical representation of the document and similarity calculation. These approaches are quite straightforward and effective but do not consider the actual meaning and context of the data, especially for complex domains like legal documents. Clustering approaches like K-Means and methods like Scatter-Gather have been implemented for effective clustering of the document into meaningful clusters.

These approaches help in effectively organizing the collection of text data but consider only the surface-level features of the data. Libraries like Scikit-learn have provided an implementation of the methods for effective application of machine learning approaches for document clustering and preprocessing. Text mining research has also been performed for classification and clustering approaches for effective extraction of meaningful patterns from the unstructured data. However, most traditional approaches consider frequency-based representation, which is not effective for extracting the context's actual meaning.

In order to avoid such limitations, embedding-based approaches have been developed. In this regard, models such as BERT use contextual embeddings to represent text meaning more accurately compared to conventional approaches. Considering such findings, the proposed system utilizes BERT embeddings along with cosine similarity to achieve semantic similarity. This helps in achieving a more accurate clustering of contracts. This results in clustering based on the context of the contract rather than keywords. This approach helps in finding relationships between documents that may not be easily obtained through conventional approaches. Moreover, the system helps in better organizing documents, which results in a more efficient information retrieval system. This helps in achieving more accurate clustering results, which may be used in various applications such as contract classification, contract compliance, etc.

2.2 Existing System

The existing systems for contract document management mainly rely on keyword search and other traditional methods of text processing. Techniques like TF- IDF and other vector space models are mainly used for document processing and similarity

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

calculations. For document clustering, methods like K-Means and hierarchical clustering are mainly used, which rely on the frequency of words in documents. These methods are quite simple and efficient in processing documents but do not take into account the context of the document.

This results in incorrect document clustering if documents with similar intentions but using different words are not grouped together. Also, in the existing systems, there is no provision for automatic labeling of clusters, which becomes a problem for the user in understanding the results of document clustering. Overall, these systems have limitations in handling semantic relationships, which make them less effective in contract analysis and management.

2.3 Proposed System

The proposed system is focused on enhancing the way in which the documents are organized based on the actual meaning of the documents rather than the keywords used in them. The proposed system is expected to manage unstructured legal documents more effectively using Natural Language Processing. In this system, the contract documents are first processed using a series of basic preprocessing operations to clean them and make them ready for use. After that, BERT is used to convert each document into a meaningful numerical form known as embeddings that represent the context of the document. Finally, these embeddings are compared using cosine similarity to determine the degree of similarity between the documents. The documents are grouped into clusters based on the similarity of the documents, even if the words used in them are completely different. In order to make the results more interpretable, the system also assigns a simple but meaningful name to each cluster. This will enable users to easily identify what type of contract is included in a certain cluster. Overall, the proposed system provides a more accurate and efficient way of organizing, searching, and managing contract documents compared to traditional approaches.

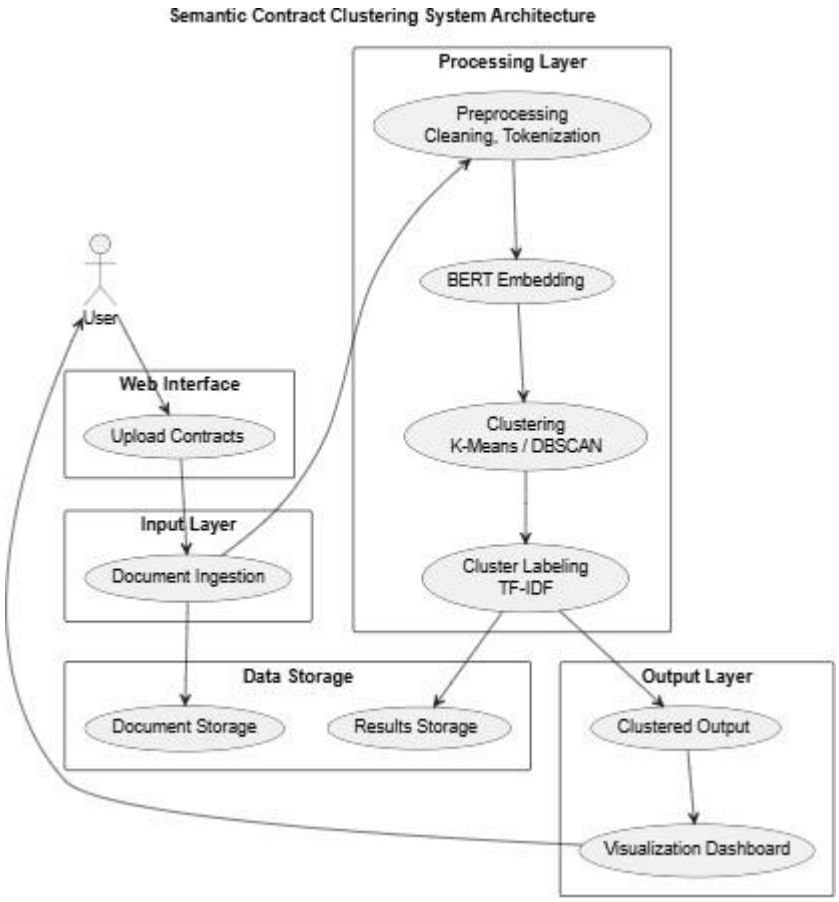


Fig 2.3.1 proposed system

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

3. SYSTEM ANALYSIS

3.1 Functional Requirements

Upload Contract Documents

The system should have the functionality for users to upload contract documents in various formats like TXT and PDF. It should also be able to upload single and multiple documents at a time. It should validate the file formats and store the documents securely. It should also have the functionality of storing the file name, upload date, and document size for future management.

Process Text Data (Pre-processing)

The system should pre-process the uploaded documents to make the text suitable for analysis. This includes:

- Converting PDF files to plain text
- Removing stop words, punctuation, and special characters
- Normalizing text (converting to lowercase)
- Tokenization (splitting text into words or sentences)
- Removing redundant information

The pre-processing step ensures that only relevant information is fed to the next step.

Generate Embeddings using BERT

The system should convert the pre-processed text into numerical vector representations using BERT (Bidirectional Encoder Representations from Transformers). This is because text embeddings using only words do not capture the meaning of text. The system should support:

- Sentence-level and document-level text embeddings
- Efficient processing of large amounts of text data
- Storage of text embeddings for faster computation

Compute Similarity

The system should be able to compute the similarity between the document embeddings by utilizing the cosine similarity function. This would be helpful in identifying the closeness between two documents in terms of their meaning or semantics. The system should be able to:

- Generate the similarity scores between the documents
- Store the similarity matrices for clustering the documents
- Enable filtering by threshold for the level of similarity

Cluster Similar Documents

The system should be able to cluster the similar documents together by utilizing the clustering algorithms such as K-Means or Hierarchical Clustering. The system should be able to:

- Determine the number of clusters or allow the user to define the number of clusters
- Group the documents that are similar in context and semantics

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Generate Labels for the Cluster

The system should be able to automatically provide meaningful labels for the clusters generated. This could be done by applying keyword extraction or summarization techniques. The labels should be:

- Meaningful representation of the overall topic of the cluster
- Easily understandable by the user
- Useful in quickly identifying the grouped documents

Display the Results

The system should be able to display the clusters and the documents in the clusters in a user-friendly manner. Some of the features that could be included are:

- Viewing the clusters and the documents in the clusters
- Highlighting key terms or summaries

3.2 Non-Functional Requirements

Performance

The system needs to provide high performance in processing and clustering the contract documents. It has to:

- Process a number of documents in a reasonable amount of time
- Use optimized models to efficiently compute the embeddings
- Compute the similarity scores without causing a delay in the system
- Provide a fast response in uploading, processing, and displaying the results

Scalability

The system should be able to scale with the increase in volume of data and users. It should be able to:

- Accommodate more contract documents with no performance compromise
- Process large volumes of data with efficiency
- Possibly accommodate cloud or distributed computing in the future
- Ensure speed and accuracy with the increase in volume

Usability

The system should have a smooth and easy user experience. This implies that the system should:

- Have a simple and clean interface
- Be easy to use with minimal technical knowledge
- Have instructions on how documents are uploaded and processed
- Have easy navigation between various sections of the system
- Have tools and guidance where necessary

Reliability

The system must be able to give good and reliable results. It must:

- Repeat the results of clustering of the same input data.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

- Be in constant operation without having any sudden failures.
- Stabilize when working on long processes.

Security

The system should ensure the safety and confidentiality of uploaded contract data. It must:

- Protect sensitive documents from unauthorized access
- Implement secure file upload mechanisms
- Ensure data encryption during storage and transmission (if applicable)
- Prevent malicious file uploads

Compatibility

The system should work across different environments. It must:

- Be compatible with major web browsers (Chrome, Edge, Firefox)
- Work on different operating systems (Windows, Linux)
- Adapt to different screen sizes (basic responsiveness)

3.3 Interface Requirements

3.3.1 User Requirements

1. Browser-Based Access

The system should be browser-accessible without the need for any additional software. It should:

- Support modern browsers such as Google Chrome and Edge
- Be platform-independent in terms of browser performance
- Function with an internet connection only

2. Simple Document Upload

The system should allow easy uploading of contract documents. It must:

- Support single and multiple file uploads
- Accept formats like TXT and PDF
- Provide drag-and-drop or file selection options
- Display upload progress and confirmation

3. Easy User Interaction

The interface should be easy to understand and user-friendly. It should:

- Have navigation menus and buttons that are easy to understand
- Have easy-to-understand instructions in the interface
- Minimize the effort needed by the user to perform tasks
- Not have complicated settings

4. Accurate Clustering Output

The system should be able to generate meaningful clustering output. For this purpose, it has to be able to:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

- Group documents based on semantic similarity
- Achieve high accuracy in clustering
- Provide confidence or similarity scores

5. Clear Result Display

The system should be able to display the output in a manner that is easily understandable. For this purpose, it has to be able to:

- Display clusters in a proper manner, including labels
- Display documents within each cluster
- Display important keywords or a summary

6. Fast Response Time

The system should be responsive to the user's actions by responding quickly.

It should:

- Reduce the waiting time before processing the request
- Display a loading icon before the processing starts
- Display the results quickly after processing the request

7. Error Handling and Notifications

The system should be able to effectively manage errors that may occur in the system. It should:

- Display error messages for invalid input
- Notify the user of failed uploads or processing errors
- Provide suggestions on how to solve the problem (e.g., "Upload a valid PDF file")
- Prevent the system from crashing due to the user's error

9. Accessibility

The system should be accessible to a wide range of users. The system should:

- Use clear fonts and good contrast
- Be simple for new users to use
- Not use complex UI components

3.3.2. System Requirements

3.3.2.1 Hardware Requirements

System: Minimum Intel Core i3 processor. Intel Core i5 processor is recommended for better performance.

Hard Disk: At least 40 GB of free space for storing documents and processing data. RAM: Minimum 4 GB. 8 GB RAM is recommended for efficient handling of BERT embeddings.

3.3.2.2 Software Requirements

- Operating System: Windows 7 or later versions, or any Linux distribution that supports Python.
- Programming Language: Python 3.x.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

- Libraries: Transformers for BERT embeddings, Scikit-learn, NumPy, Pandas, Flask.
- Framework: The Flask web development framework.
- Browser: Any web browser that supports running web applications (for example, Chrome or Edge).

3.3.3 Performance Requirements

1. Document Ingestion and Preprocessing

- The system should efficiently extract and preprocess text from PDF, TXT, and DOCX files.
- It must remove noise such as headers, footers, and special characters while preserving meaningful content. Processing should be fast and support multiple documents.

2. Embedding Generation and Similarity

- The system should be able to generate accurate semantic embeddings with BERT models.
- It should also be able to compute cosine similarity efficiently, especially with large numbers of documents, to avoid any form of delay.

3. Clustering Performance

- The system should be able to automatically cluster similar documents, like employment contracts, NDAs, or leases.
- The accuracy, significance, and speed of the cluster formation are also important.

4. Cluster Naming and Interpretability

The system should be able to provide meaningful names to the clusters in terms of the presence of important keywords or patterns, making it easy to interpret the results.

5. Scalability and Throughput

The system should be able to efficiently process a large number of documents without compromising the efficiency of the system.

6. Search and Retrieval

The system should be able to search and retrieve similar documents or clauses in less than 1 second.

7. Error Handling and Robustness

The system should be able to handle invalid or corrupted files in a smooth manner, without any interruption in processing.

8. Visual Representation

The system should be able to offer a good visual representation of clusters, document count, and terms, even for large sets of data.

9. Data Security

The system should be able to ensure a good level of security for the data, including encryption, without violating fundamental legal principles.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

4. SYSTEM DESIGN

4.1 Design Approach

1. User Interface Design

The design of the system is such that it incorporates a simple and interactive user interface. The user is able to upload a single or multiple contract documents. The system also provides a search facility to efficiently retrieve the required information from the clustered results.

2. Data Preprocessing

The data is preprocessed by cleaning and tokenizing the data and removing stop words. This stage is important to refine the data for further analysis.

3. Text Representation (Embedding Generation)

The preprocessed data is converted into a vector form by applying Natural Language Processing techniques. The vectors are able to comprehend the meaning of the data for efficient comparison.

4. Clustering Mechanism

The clustering algorithm is utilized to perform clustering operations on similar documents based on semantic similarity. This aids in clustering contracts in a meaningful manner, as opposed to simple keyword-based clustering.

5. Cluster Labeling

A meaningful label is generated for each cluster based on frequent terms and patterns identified within the documents contained within a cluster.

6. Result Visualization and Search

The clustered results are displayed to the user in an organized manner through the interface, allowing for semantic search to be performed on the result

4.2 UML Diagrams

Unified Modeling Language (UML) diagrams are a standardized way of visualizing the design and architecture of software systems. UML provides a universal language for software developers to plan, model, and document systems in an organized manner. Instead of relying solely on code to communicate design, UML diagrams allow teams to graphically represent both the structure and behavior of the system being built. This helps in understanding complex software architecture and improves communication between developers, analysts, stakeholders, and clients. UML was developed and is maintained by the Object Management Group (OMG), a consortium that sets standards in software development. UML is not a programming language but rather a visual language that supports system design and analysis across various stages of the software development life cycle. At its foundation, UML consists of two major components: a meta-model and a notation system. The meta- model defines the language's core structure and rules, while the notation represents these concepts graphically in diagrams such as class diagrams, use case diagrams, and sequence diagrams. Over time, UML has evolved to potentially include additional elements like methodologies or development processes, making it a flexible and extensible tool that adapts to different types of modeling requirements beyond software—such as business processes and workflows.

Purpose of UML

- To specify the structure and behavior of software systems clearly and unambiguously.
- To visualize system architecture through standardized diagrams, aiding quick understanding of complex designs.
- To document software artifacts for easier maintenance, enhancement, and scalability over time.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

- To construct models that act as blueprints for implementation, testing, and deployment.
- To serve as a communication tool among analysts, developers, testers, and clients, reducing misunderstandings in system design.

Goals of UML

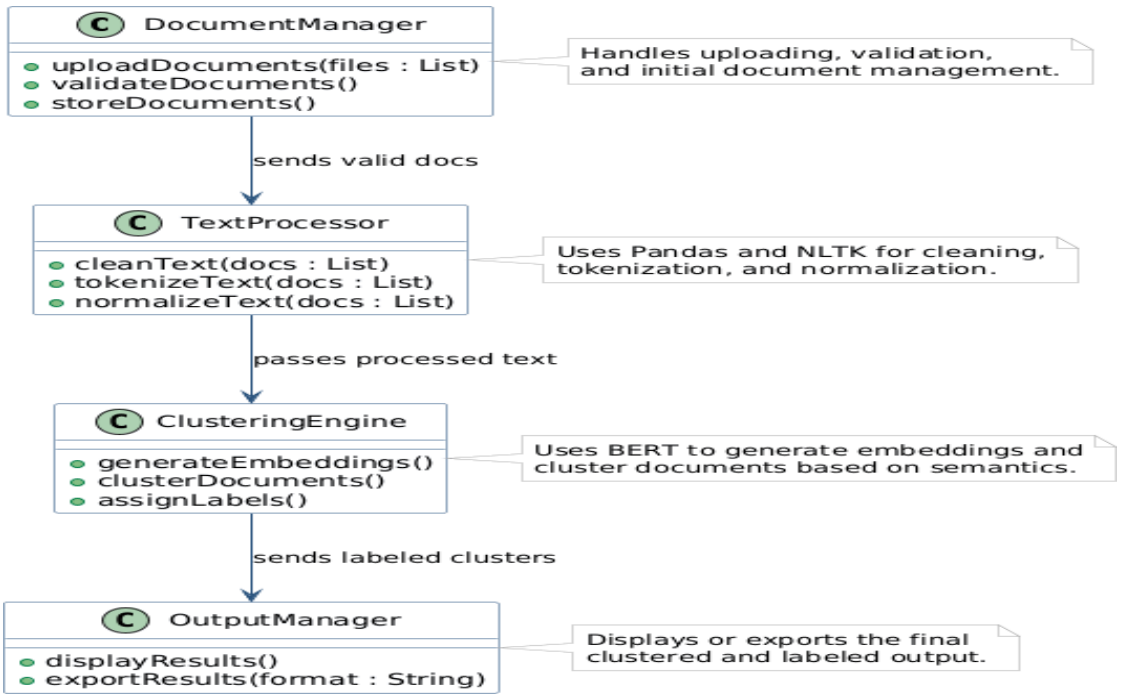
- Provide a standard visual language for meaningful modeling.
  - Be independent of programming languages and processes.
  - Promote design reuse through patterns and components.

4.2.1 Class Diagram for User

A class diagram in UML is a type of static structure diagram that represents the classes within a system. It details each class's attributes and operations (or methods) and illustrates the relationships among these classes. By showing how classes are connected, the diagram clarifies how information is organized and managed throughout the system. This makes it easier to understand the system's overall structure and design.

The class diagram represents the architecture of the Semantic-Based Contract to Document Clustering System. It includes a **DocumentManager** module that uploads, validates, and stores contracts from various formats. The **TextProcessor** module cleans, tokenizes, and normalizes the text using Pandas and NLTK. The **ClusteringEngine** generates BERT embeddings, performs semantic clustering, and assigns meaningful labels to each cluster. Finally, the **OutputManager** displays or exports the clustered and labeled results for users.

Fig 4.2.1 class diagram for user



International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

4.2.2 Use Case Diagram for User

A Use Case Diagram in UML is a behavioral diagram that visually represents the interactions between actors (external users or systems) and the system's functionalities (use cases). It shows what functions are performed and who initiates them. Use cases depict specific goals, while the diagram also illustrates relationships and dependencies among them. This helps clarify the system's overall behavior and user roles.

A Use Case Diagram in UML is a behavioral diagram that shows the interaction between users and the functionalities of the Semantic-Based Contract Document Clustering System. It demonstrates how a user uploads contracts using an interface, while the system is performing operations such as pre-processing text, generating BERT embeddings, clustering based on cosine similarity, and providing meaningful labels to clusters. It shows the primary functionalities of the system and the interaction between the user and the clustering platform.

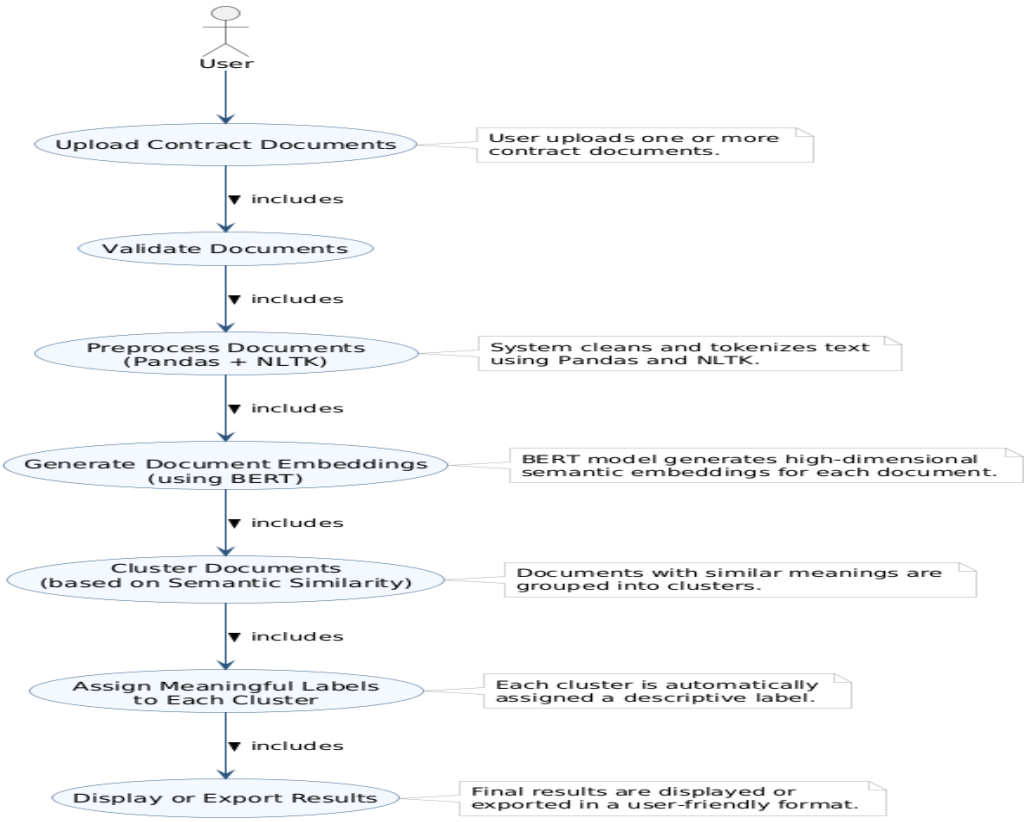


Fig 4.2.2 use case diagram for user

4.2.3 Sequence Diagram for User

A sequence diagram in UML is a type of interaction diagram that illustrates how processes interact with each other and the order of these interactions. It is a form of Message Sequence Chart. Sequence diagrams are also known as event diagrams, event scenarios, or timing diagrams. They help visualize the flow of messages between different parts of a system over time.

A Sequence Diagram in UML is an interaction diagram that illustrates how processes interact with each other and the order of these interactions within the Semantic-Based Contract Document Clustering System. It is a form of Message Sequence Chart, commonly known as event diagrams, event scenarios, or timing diagrams. These diagrams help visualize the flow of messages between different system components over time, from user contract uploads through BERT embedding generation, cosine similarity clustering, to result visualization.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

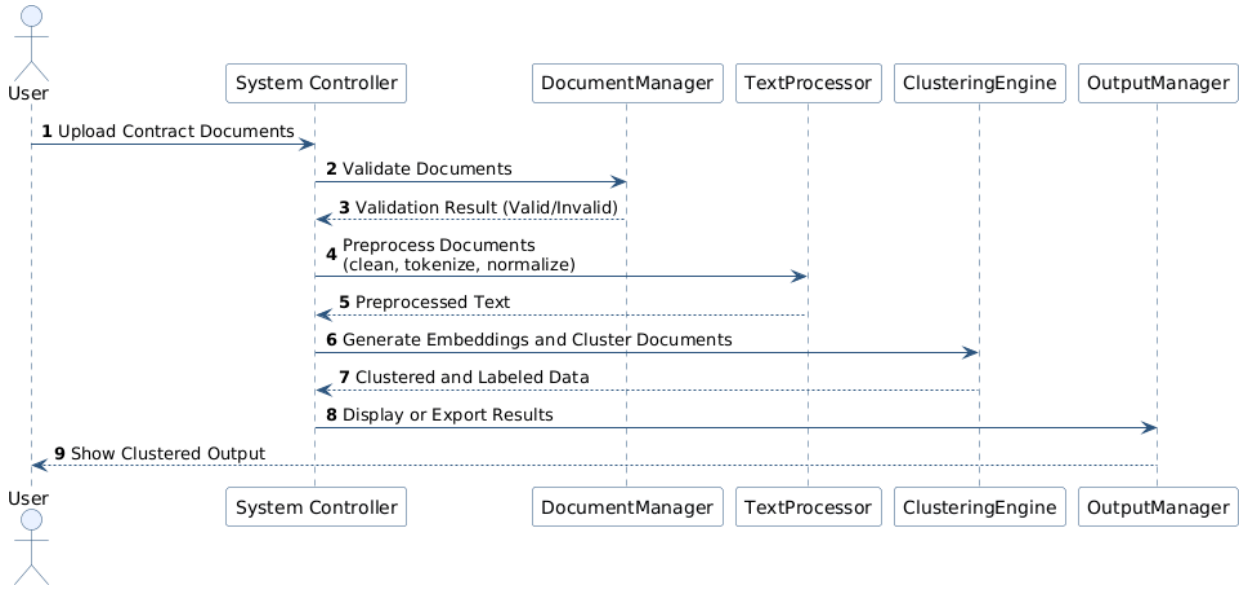


Fig 4.2.3 sequence diagram for user

4.2.4 Data Flow Diagram for user

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. There are 2 levels in the below diagram.

The first diagram, a Level 0 DFD, provides a high-level overview of the Semantic-Based Contract Clustering System, where external users such as the Admin and Legal Team interact with the system. Users upload contracts to the system and receive cluster reports and notifications. The system interacts with an external storage or database to store and retrieve contract documents. This diagram illustrates the overall flow of data between users, the system, and storage, providing a clear understanding of the system's scope and primary functionality without going into internal processing details.

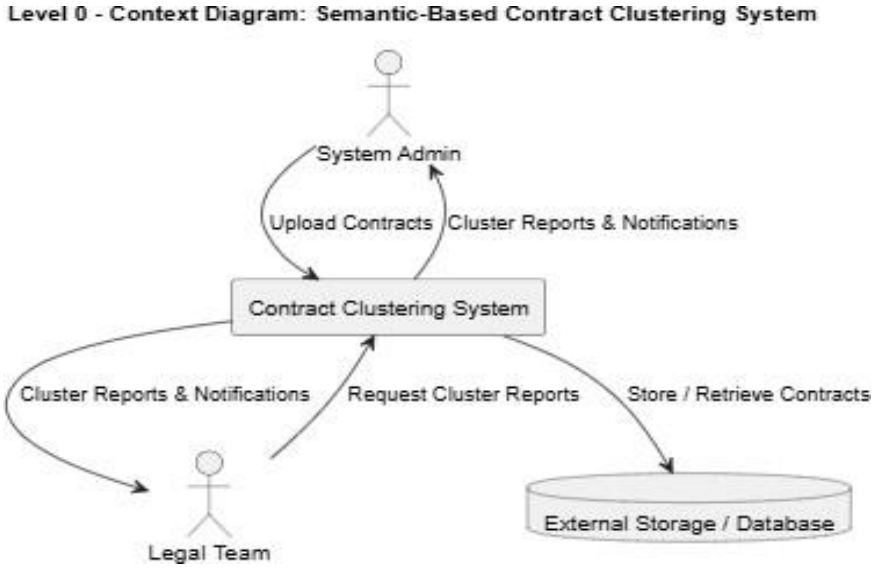


Fig 4.2.4 (a) Level-0 Data Flow Diagram

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

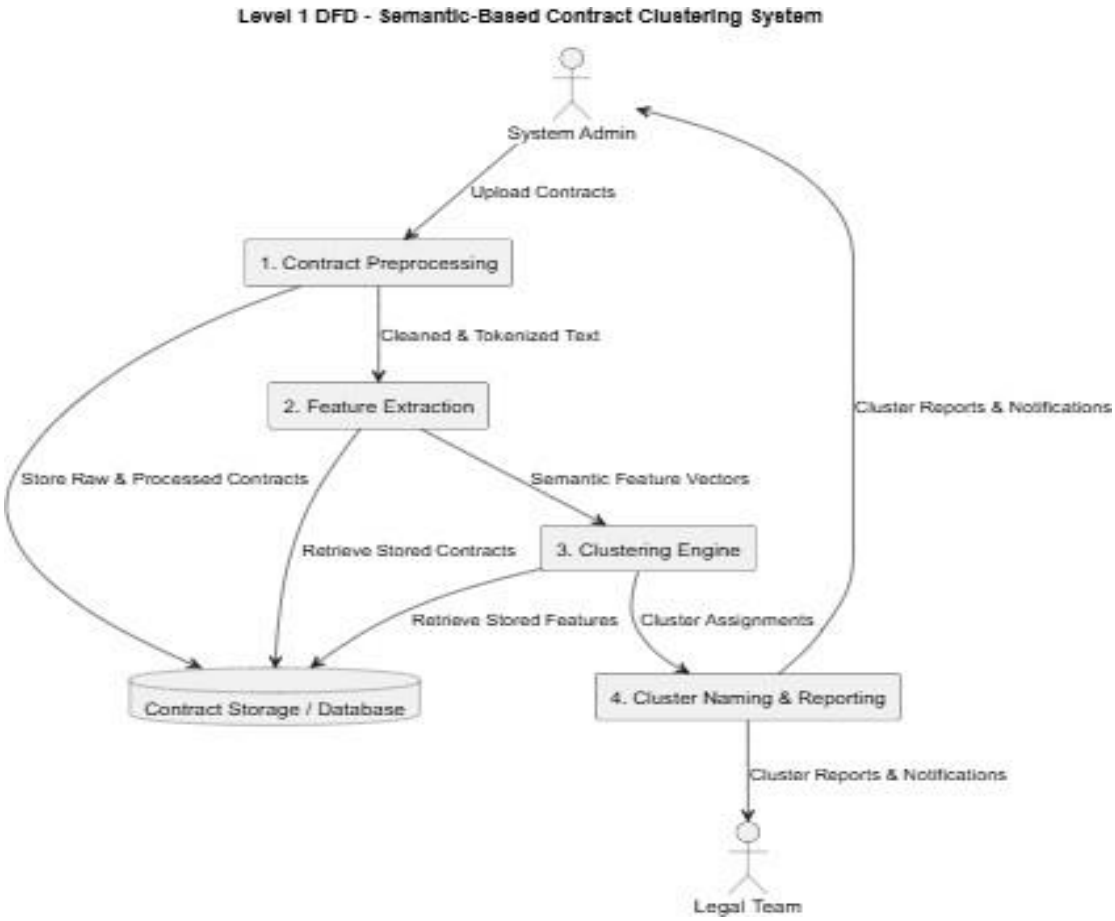


Fig 4.2.4 (b) Level-0 Data Flow Diagram

4.2.5 Activity Diagram for User

Activity diagrams are graphical representations of workflows, showing step-by-step activities and actions with support for choices, loops, and parallel processes. In UML, activity diagrams describe the business and operational workflows of system components. They illustrate the sequence and conditions for coordinating lower-level behaviors. Overall, activity diagrams visualize the flow of control throughout a process.

The Activity Diagram depicts the workflow of the Semantic-Based Contract Clustering System. The process starts with the user uploading the contracts, which are validated and then preprocessed by tokenization, stopword removal, and lemmatization. The system now performs the extraction of semantic features and clustering of the contracts and provides meaningful names for the clusters. The Admin and Legal Team receive reports and notifications, and the invalid contracts are rejected. This diagram effectively depicts the process of the contracts being organized into understandable clusters.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

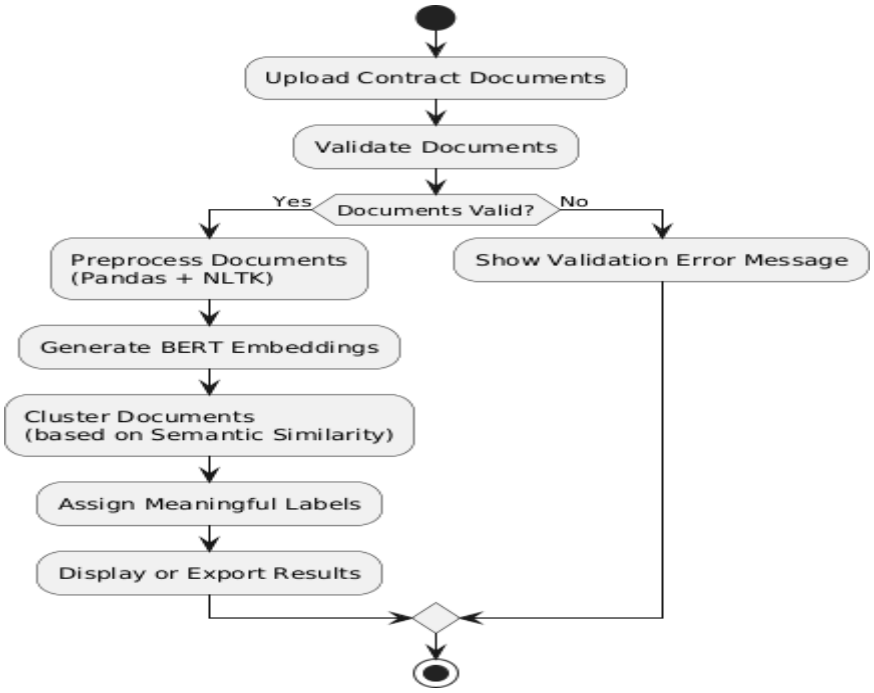


Fig 4.2.5. Activity Diagram for user

4.3 Modules

4.3.1 User Interface Module Contract

Upload and Access

Web-based Contract Upload Interface:

Users (Admin or Legal Team) can upload contracts using a web interface implemented using Flask. It supports different formats like PDF, DOCX, and TXT. It also provides an interface to fill in contract information in a user-friendly manner.

Viewing Cluster Report:

After clustering, users can view the generated cluster reports and notifications in the browser itself.

Display of Clustering Information and Metrics

Cluster Insights: Users can view cluster information like the number of contracts in the cluster, frequently used terms, and semantic similarity score.

Performance Metrics: Optional metrics to show the user the time taken to process each document and the system's status.

4.3.2 System Module

Contract Preprocessing Module

Text Cleaning: This module removes unwanted characters and formatting from the text.

Tokenization and Lemmatization: This module splits the text into words and lemmatizes the words.

Stop Word Removal: This module removes common words that do not contribute to the meaning of the sentence.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Feature Extraction Module

Semantic Vector Generation: This module uses the preprocessed text and generates feature vectors with the help of NLP models such as BERT and TF-IDF

Feature Storage: This module stores the generated features in the system database.

Clustering Engine Module

Similarity Calculation: This module calculates the similarity between contracts**. Cluster Formation:** This module forms clusters based on the similarity score.

Cluster Validation: This module validates the clusters to ensure consistency**. Cluster Naming and Reporting Module**

Frequent Term Analysis: This module analyzes the frequent terms present in the clusters**.**

Cluster Naming: This module provides meaningful names for the clusters**.**

Report Generation: This module generates downloadable reports for the Admin and Legal Team.

4.3.3 End User Module

Contract Access and Cluster Viewing

Web-Based Access: End users can access uploaded contracts, cluster membership, and cluster reports using a standard web browser**.**

Interactive Reports: Users can click on clusters to view individual contracts and their semantic information.

System Feedback and Monitoring

Processing Feedback: Users are provided with notifications upon successful uploading, processing, and clustering of contracts.

5. IMPLEMENTATION AND RESULTS

5.1 Method of Implementation

The proposed Semantic Contract Clustering System is written in Python with the help of advanced Natural Language Processing (NLP) methods, specifically with the usage of the BERT (Bidirectional Encoder Representations from Transformers) model. The main purpose of the system is to automatically group contracts in accordance with their semantic similarity and hence minimize the manpower, enhance the efficiency of document search and offer insights of meaning to the Admins and Legal Teams. The system is based on the modular architecture in which each module executes a certain function individually. The document ingestion, preprocessing, feature extraction with BERT, similarity computation, clustering, cluster naming and reporting are the important modules.

To begin with, contract documents can be uploaded in various formats through the web-based interface of the system, like PDF, DOCX, and TXT. Such documents are then sent to preprocessing module, where the text content is then removed and purged. It is preprocessed by first removing noise like special characters, header, and footer after which it is then processed by tokenizing, removing stopwords and lemmatizing. Cleaning of the text is then followed by sending the cleaned text to the BERT feature extraction module.

During this step, the text is turned into thick representations of vectors called embeddings. These embeddings obtain the contextual and semantic meaning of the contract text, and thus, they are very useful in similarity analysis. The obtained embeddings are saved at the system database to be processed and used again. The clustering module then calculates the similarity between the contract embeddings with the cosine similarity and clusters similar contracts with clustering algorithms like the K-

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Means algorithm. After the process of clustering, the system uses a labeling mechanism to come up with meaningful labels of each cluster. This is through the analysis of common terms and pattern of clauses that occur in each cluster. Lastly, the analysis is presented in the form of a web interface, where the user would see clustered documents, browse and individual contracts, and download reports. This system also measures performance metrics like processing time, cluster size, similarity scores to measure its efficiency. The system is scalable, flexible and can be easily maintained because of the modular design, which makes it applicable to the real world applications of contract management.

5.1.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is the branch of artificial intelligence that deals with the interaction between computers and human language. It allows computers to process, analyze, and understand human language, thus helping them communicate more efficiently.

NLP plays an important part in the automation of tasks dealing with unstructured data, such as legal contracts, emails, and reports. Contract reports, for instance, are often written in complex legal language, but NLP helps transform this unstructured data into structured data, thus making it machine-readable.

Core NLP Techniques

- The NLP techniques utilized in this project include the following:
- Text Extraction: The raw text from PDF, DOCX, or TXT files will be extracted.
- Tokenization: The raw text will be split into sentences or words.
- Stopword Removal: The raw text will be cleaned up by removing unnecessary words such as “the”, “is”, “and”, etc.
- Lemmatization: The raw text will be converted into its base form, such as “running” becoming “run.”
- Text Normalization: The raw text will be converted into lower case, removing special characters.
- Feature Preparation**:** Preparing clean text for embedding generation

Role of NLP in This Project

The role of NLP in the Semantic Contract Clustering System is the foundation layer, where it prepares the contract data for analysis.

The functions of NLP include:

- Cleaning Contract Data: It removes unnecessary data from the contracts
- Handling Legal Language: It processes legal language effectively
- Improving the Quality of the Data: It ensures only necessary data is passed to the BERT model
- Standardizing the Input: It makes the contracts uniform, irrespective of their

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

format

- Helping in the Correct Clustering: It ensures the data is clean, resulting in correct clustering
- For instance, two contracts might be worded differently, but their meanings might be the same. The NLP will ensure the unnecessary variations are removed.

Importance of NLP in Contract Analysis

- It converts unstructured legal documents into a data form for analysis.
- It reduces manual effort required to read and classify contracts.
- It increases the accuracy of subsequent steps such as clustering.
- It automates the process of managing legal documents.

5.1.2 BERT (Bidirectional Encoder Representations from Transformers)

BERT is a highly effective deep learning technique developed by Google in 2018. It is a transformer-based model. BERT uses the transformer model to understand the contextual meaning of a word by considering the context from both sides of the word.

BERT differs from traditional NLP approaches that use keyword matching or frequency-based approaches. BERT uses the semantic meaning of the text, making it highly effective for analyzing complex contracts.

Key Features of BERT

Bidirectional Context Understanding:

BERT reads the text from both sides of the word to understand the complete context of the word.

Pre-trained Model:

BERT has been trained on large datasets like Wikipedia.

Semantic Embeddings:

BERT converts text into numerical representations of meaning.

Fine-tuning Capability:

BERT can be fine-tuned on domain-specific data sets like contracts.

Clause Awareness:

BERT can be used to identify clauses like termination, confidentiality, and liability clauses.

Robustness:

BERT is highly effective in handling variations of language, synonyms, and writing styles.

Working of BERT in this Project

The working of BERT in the Semantic Contract Clustering System is as follows:

- **Input Processing:** The preprocessed input contracts are passed into the BERT model.
- **Token Encoding:** The input text is transformed into tokens that BERT can process.
- **Embedding Generation:** BERT creates embeddings of contracts.
- **Semantic Representation:** The embeddings represent the semantic meaning of contracts or clauses.

Role of BERT in This Project

BERT plays a vital role in the system by providing a semantic understanding of contracts:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Captures Meaning, Not Just Words:

BERT clusters contracts with different wording but similar meaning.

Supports Similarity Computation:

BERT facilitates the calculation of cosine similarity between contracts.

Enhances Cluster Naming:

BERT provides significant words from the embeddings for better naming of clusters.

Handles Legal Complexity:

BERT has a semantic understanding of context-dependent legal terms and clauses.

Advantages of Using BERT in This System

- High accuracy in semantic understanding
- Better handling of legal language
- Reduces dependency on manual feature engineering
- Scalable for large document datasets
- Improves overall system performance

5.1.3 What is Python

Python is a high-level programming language that is easy to learn and can be used for artificial intelligence, data analysis, and web development. It can be considered the most suitable programming language for building NLP-based systems such as the proposed Semantic Contract Clustering System. In the proposed system, Python acts as the main programming language for preprocessing the contracts, creating BERT embeddings, and clustering, as well as creating a web interface for the system. Python provides an easy way to integrate NLP, machine learning, and web technologies in one environment.

Rich Library Support:

Access to rich libraries that support the requirements of NLP and ML, including:

- **transformers** for BERT embeddings
- **scikit**-learn for clustering and similarity calculations
- **numpy** and pandas for handling and manipulating data
- **nltk** or spacy for text preprocessing
- **Flask** for creating the web interface

Simple Syntax and Readability: The syntax of Python makes development easier, saving time in creating the code. It also makes code easier to read and debug.

Cross-Platform Compatibility: The code written in Python can run seamlessly on Windows, Linux, and macOS, ensuring the platform's portability.

Rapid Prototyping and Integration: The flexibility of Python allows for rapid prototyping and integration of different clustering algorithms and BERT.

Data Handling and Numerical Computing: With the help of numpy and pandas libraries, Python can efficiently handle large volumes of data related to contracts. It also efficiently performs operations and stores semantic embeddings.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Web Integration: Using the Flask framework, Python can be used to develop a simple web application for uploading contracts and viewing reports for clusters in real time. **Support for NLP and AI Workflows**: The Python libraries help in efficiently performing operations such as tokenization, feature extraction, and computing semantic similarity and clustering, thereby making it an end-to-end solution for contract management.

5.1.4 Python Modules and BERT Implementation

The Semantic Contract Clustering System is implemented in Python, which is considered to be the backbone of this project, integrating all the components of NLP, BERT, clustering, and web reporting together. Some of the key components of this system and their purposes are:

Transformers (Hugging Face):

This library is utilized to implement BERT to obtain semantic embeddings of contracts. Each contract is embedded in a dense vector space, where the meaning of all clauses and sentences is represented. This is considered to be the basis of clustering similar contracts together.

Scikit-learn:

This library is utilized to implement clustering algorithms such as K-Means or Hierarchical Clustering, which takes the embeddings of contracts and groups them based on their similarity in content.

NumPy and Pandas:

NumPy is utilized to handle numerical operations on BERT embeddings, while pandas is utilized to handle contract metadata, clustering, and performance in a tabular format.

NLTK/spacy:

These libraries are employed for preprocessing the contract data. They are used to tokenize the sentences and words, remove stop words, and perform lemmatization. This preprocessing step ensures that the BERT model receives clean data to compute accurate embeddings.

Flask:

This framework is employed to create a web-based interface through which users can upload the contracts, view the clustering results, and download reports.

Python combines the web interface with the backend clustering and BERT.

Pickle:

The Pickle module in Python is employed to serialize and save the BERT embeddings or models for future use. This prevents the calculation of embeddings for previously processed contracts.

Time and Logging:

The time and logging modules in Python are employed to monitor the time consumed by the preprocessing step, embedding generation, and clustering. These modules also log system operations for debugging and performance evaluation.

Workflow for BERT Implementation:

The Python workflow for BERT implementation includes the following steps: contracts are uploaded and preprocessed, BERT computes the embeddings, clustering algorithms are employed to group similar contracts, and the most frequently occurring words are analyzed to provide meaningful names to the clusters.

5.1.5 Setup and Installation

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

The Semantic Contract Clustering System is implemented entirely in Python and relies on various libraries for NLP, BERT embeddings, clustering, and web interface. The setup process begins with installing Python and proceeds through library installation, model downloads, and running the system.

Step 1:

Install Python The system requires **Python 3.8 or later**. Python can be downloaded from the official Python website. After installation, verify it by running the command:

python –version

This ensures that Python is installed correctly and is available in your system path.

Step 2: Create a Virtual Environment

It is recommended to create a virtual environment to isolate project dependencies. Run the following command to create one:

python -m venv contract_env

Activate the environment:

- On Windows: contract_env\Scripts\activate
- On Linux/macOS: source contract_env/bin/activate

This ensures that all libraries installed for this project do not interfere with other Python projects.

Step 3: Install Required Python Libraries

The project depends on several Python libraries for NLP, BERT embeddings, clustering, and web interface. Install them using pip:

pip install transformers scikit-learn numpy pandas nltk spacy flask pickle-mixin

Step 4: Download NLP Models

For text preprocessing, download required models for NLTK and spaCy. Using NLTK:

import nltk nltk.download('punkt') nltk.download('stopwords')

For spaCy, download the English language model:

import spacy

nlp = spacy.load('en_core_web_sm')

These models enable tokenization, stop-word removal, and lemmatization for contract text.

Step 5: Download Pre-trained BERT Model

The system uses the Hugging Face transformers library to access BERT. The pre-trained model can be downloaded with:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased') model = BertModel.from_pretrained('bert-base-uncased')

This model converts contract text into semantic embeddings that capture the meaning of clauses and sentences. Fine-tuning the model on contract-specific datasets is optional but can improve clustering accuracy.

Step 6: Run the Web Interface

Flask is used to provide a web interface for uploading contracts and viewing cluster results. To start the interface, run:

python app.py

Open a browser and navigate to http://127.0.0.1:5000 to access the system. Users can upload contracts, view clustering results, and download reports directly from the web interface.

Step 7: Test the System

After setup, test the system by uploading sample contracts. The system should preprocess the text, generate BERT embeddings, cluster the contracts, assign meaningful cluster names, and display the results.

5.2 Method of Implementation:

Frontend Interface:

```
<!DOCTYPE html>
<html>
<head>
<title>DocCluster AI</title>
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>

<body>

<nav class="navbar navbar-dark bg-dark px-4">
<h3 class="text-white">DocCluster AI</h3>
</nav>

<div class="container text-center text-white mt-5">

<h1>AI Document Clustering</h1>

<form action="/cluster" method="post" enctype="multipart/form-data">
<div class="upload-box">
<input type="file" name="files" multiple class="form-control">
<button class="btn btn-primary mt-3">Upload & Cluster</button>
</div>
</form>

</div>
</body>

</html>
```

Explanation:

The index.html file serves as the frontend interface of the system, designed using standard HTML along with Bootstrap for styling and responsiveness, and integrated with Flask using the url_for function to link static resources like CSS; the page includes a navigation bar displaying the application name “DocCluster AI” and a centered user interface where users can upload

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

multiple documents through a file input field, while a form is configured to send these files to the backend /cluster route using the POST method with multipart/form-data encoding to support file uploads, and upon clicking the “Upload & Cluster” button, the selected documents are transmitted to the server for processing, making this page the entry point of the system that enables user interaction and initiates the document clustering workflow.

Result.html

```
<!DOCTYPE html>
<html>
<head>
<title>Clusters</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>

<body class="bg-dark text-white">

<div class="container mt-5">

<h2> AI Clustering Result</h2>

<!--  SEARCH -->
<form action="/search" method="post" class="d-flex mb-4">
        <input type="text" name="query" class="form-control me-2" placeholder="Search
by meaning..." value="{{ query }}">
<button class="btn btn-warning">Search</button>
</form>

<!--  SEARCH RESULTS -->
{% if search_results %}
<h4> Results</h4>

{% for cluster, files in search_results.items() %}
<div class="card bg-success mt-3 p-3">
<h5> {{ cluster }}</h5>
<ul>

{% for file in files %}
<li>{{ file }}</li>
{% endfor %}
</ul>
</div>
{% endfor %}

<hr class="mt-5">
<h4> All Clusters</h4>
{% endif %}

<!-- ALL CLUSTERS -->
{% for cluster, files in clusters.items() %}
<div class="card bg-secondary mt-3 p-3">
<h4>{{ cluster }}</h4>
<ul>
{% for file in files %}
<li>{{ file }}</li>
{% endfor %}
</ul>
</div>
{% endfor %}
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

```
<a href="/" class="btn btn-light mt-4">Go Back</a>

</div>

</body>
</html>
```

Explanation:

The result.html file is responsible for displaying the output of the document clustering system in a structured and user-friendly manner, built using HTML and styled with Bootstrap to provide a responsive and visually clear interface, while dynamic content rendering is handled through Flask using Jinja templating; the page includes a search feature where users can enter a query to find semantically similar documents, and upon submission, the query is sent to the /search route for processing, with results displayed in highlighted cards showing relevant clusters and their associated files, and below this, all clusters generated by the system are displayed in separate sections, each labeled with a cluster name and listing the documents it contains, allowing users to easily explore grouped contracts, while a “Go Back” button enables navigation to the main upload page, making this interface an essential component for visualizing clustering results, supporting semantic search, and improving overall document accessibility and analysis.

Backend Interface:

```
from flask import Flask, render_template, request import os
import re
import pandas as pd
from collections import Counter

from sklearn.cluster import AgglomerativeClustering
from sentence_transformers import SentenceTransformer, util app = Flask(   name   )

UPLOAD_FOLDER = "uploads" os.makedirs(UPLOAD_FOLDER, exist_ok=True)


GLOBAL_CLUSTERS = {} GLOBAL_DOCS = {}

model = SentenceTransformer('all-MiniLM-L6-v2') # -------- FILE READER      #
def read_file_content(path): text = ""
try:
if path.endswith(".txt"):
text = open(path, encoding="latin-1").read()

elif path.endswith((".xlsx", ".xls", ".xlsm")): excel_file = pd.ExcelFile(path)
all_text = []

for sheet in excel_file.sheet_names: df = excel_file.parse(sheet)

cols = " ".join([str(c) for c in df.columns]) values = " ".join(df.astype(str).values.flatten())

    all_text.append(cols + " " + values) text = " ".join(all_text)

elif path.endswith(".csv"): df = pd.read_csv(path)
        text = " ".join(df.astype(str).values.flatten()) except Exception as e:




print("Error reading:", path, e) text = ""

return text.lower()
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

```
# -------- CLUSTER NAME------------------------ #
def get_cluster_name(texts): full_text = " ".join(texts)

words = re.findall(r"\b[a-zA-Z]{3,}\b", full_text)

stopwords = [ "the","and","is","are","was","were",
"this","that","for","with","you","your", "have","has","had"
]

words = [w for w in words if w not in stopwords] common = Counter(words).most_common(3)

if common:
    return common[0][0].capitalize() else:
return "General"
#                    HOME _ _ .          #    ---------
@app.route("/") def home():
return render_template("index.html")


# -------- CLUSTER ----------------------#
@app.route("/cluster", methods=["POST"]) def cluster():

files = request.files.getlist("files") documents = []
filenames = [] for file in files:

if file.filename == "": continue


path = os.path.join(UPLOAD_FOLDER, file.filename) file.save(path)

text = read_file_content(path) if text.strip() != "":
documents.append(text) filenames.append(file.filename)

if len(documents) == 0:
return " No valid content found!"

# SEMANTIC EMBEDDINGS
embeddings = model.encode(documents) k = min(5, len(documents))

clustering = AgglomerativeClustering(n_clusters=k) labels = clustering.fit_predict(embeddings)

clusters = {} cluster_texts = {}

for i, label in enumerate(labels): clusters.setdefault(label, []).append(filenames[i])
cluster_texts.setdefault(label, []).append(documents[i])

# AUTO NAMING
named_clusters = {}

for label in clusters:
name = get_cluster_name(cluster_texts[label]) named_clusters[name] = clusters[label]

# SAVE FOR SEARCH
global GLOBAL_CLUSTERS, GLOBAL_DOCS
GLOBAL_CLUSTERS = named_clusters GLOBAL_DOCS = {}

for name, files in named_clusters.items(): for f in files:
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

```
GLOBAL_DOCS[f] = name

    return render_template("clusters.html", clusters=named_clusters, search_results=None,
query="")


# -------- SEARCH (SAME PAGE)------------------- #
@app.route("/search", methods=["POST"])


def search():

query = request.form.get("query", "").strip() if query == "":
        return render_template("clusters.html", clusters=GLOBAL_CLUSTERS, search_results=None,
query="")

results = {}

query_embedding = model.encode(query)

for filename in os.listdir(UPLOAD_FOLDER):

path = os.path.join(UPLOAD_FOLDER, filename) text = read_file_content(path)

if text.strip() == "": continue

doc_embedding = model.encode(text)

score = util.cos_sim(query_embedding, doc_embedding) if score > 0.3:

cluster_name = GLOBAL_DOCS.get(filename, "Others") results.setdefault(cluster_name, []).append(filename)

return render_template( "clusters.html", clusters=GLOBAL_CLUSTERS, search_results=results, query=query
)

#  RUN- - - - .  # - - - - - -.
if  name  == "  main  ": app.run(debug=True)
```

Explanation:

The program is developed using Flask to build a web-based Semantic Document Clustering system that processes and organizes uploaded files using advanced Natural Language Processing techniques powered by BERT via the SentenceTransformers library; initially, users upload documents in formats such as TXT, CSV, or Excel, which are read and converted into clean text using file-handling and preprocessing functions, after which BERT generates semantic embeddings for each document to capture their contextual meaning, and these embeddings are grouped using Agglomerative Clustering from Scikit-learn to form clusters of similar documents, followed by an automatic cluster naming mechanism that extracts frequently occurring meaningful words while removing stopwords, ensuring interpretability; the system stores cluster data globally to enable efficient same-page semantic search, where user queries are also converted into embeddings and compared with document embeddings using cosine similarity to retrieve relevant files above a threshold score, and finally, all clustered results and search outputs are dynamically rendered on a web page, providing an interactive, scalable, and intelligent solution for organizing and retrieving large volumes of document.

Document.js

```
const canvas = document.getElementById("particles"); const ctx = canvas.getContext("2d");

canvas.width = window.innerWidth; canvas.height = window.innerHeight; let particles = [];

for (let i = 0; i < 80; i++) { particles.push({
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

```
x: Math.random() * canvas.width, y: Math.random() * canvas.height, r: Math.random() * 3
});
}
function draw() {
ctx.clearRect(0, 0, canvas.width, canvas.height); ctx.fillStyle = "white";

particles.forEach(p => { ctx.beginPath();
ctx.arc(p.x, p.y, p.r, 0, Math.PI * 2);

ctx.fill();
p.y += 0.5;
if (p.y > canvas.height) p.y = 0;
});

requestAnimationFrame(draw);
}
draw();
```

Explanation:

This is a simple particle animation script written in JavaScript. It utilizes the HTML5 Canvas element for creating a dynamic animation effect. First off, it selects a canvas element and establishes a 2D drawing context to start rendering the animation. The canvas is set to the entire height and width of the browser window for a full-screen effect. An array of 80 particles is then created, with each particle assigned a random x and y position along with a small radius. The draw function is then defined to start the animation process. Inside the draw function, the canvas is cleared in every frame to prevent overlapping of drawings. The particles are then rendered on the canvas as white circles using the arc method. The particles are then moved downwards slowly by incrementing their y positions. When a particle reaches the bottom of the screen, it resets to the top of the screen for a continuous effect. The requestAnimationFrame method is utilized to repeatedly call the draw function for a smooth animation effect.

Document.css

```
body {
background: linear-gradient(to right, #141e30, #243b55); color: white;
}
.upload-box {
border: 2px dashed white; padding: 40px;
border-radius: 15px;
}
```

Explanation:

The above CSS code contains the visual styles for the webpage. The body selector contains the code for the smooth linear background gradient, changing color from dark blue (#141e30) to lighter blue (#243b55), giving it a modern look. It also contains the code for changing the color of the text to white, providing high contrast for easy reading on the dark background. The class ".upload-box" contains the code for the container element, mostly for file upload. It contains the code for adding a dashed white border around the container, along with 40 pixels of padding for spacing inside the container. The border radius rounds the corners of the container, making it look smoother by 15 pixels. The above code improves the user interface by making it look neat,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

attractive, and easy to use.

5.3 Output Screens:

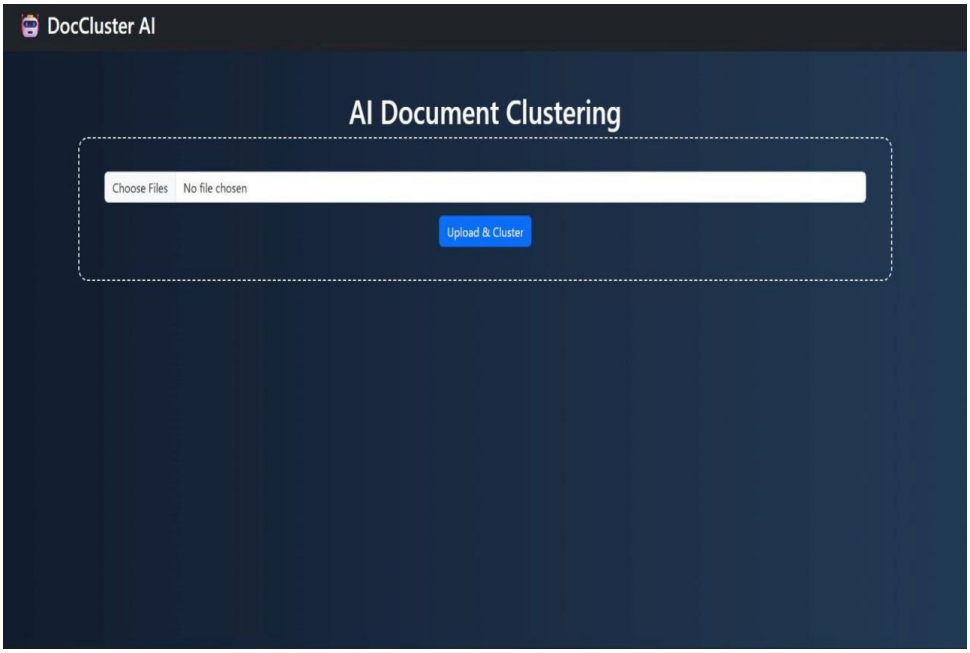


Fig 5.3.1 AI-powered document clustering tool ready for your uploads.

This interface is the first step in the system, where users can upload a single file or multiple files for clustering purposes. It is a simple interface with a file selection option and a button titled "Upload & Cluster." After a file is uploaded, the system will proceed with preprocessing, which includes cleaning and tokenization, and then proceed with generating embeddings using NLP techniques. This interface acts as a starting point for the entire clustering system, allowing for a smooth interaction with the backend system.

Fig 5.3.2: Complete view of all document clusters and categories.

This output screen indicates the result of the document clustering process, where a number of documents are grouped in a meaningful manner according to their semantic similarity. Each cluster of grouped documents is represented with a keyword or a

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

theme, such as “How,” “Apple,” “Students,” and “Education.” This helps users to comprehend the clustering process in a much clearer manner. The individual documents in a cluster are clearly represented, which helps users to comprehend their relationship with other documents in a cluster.

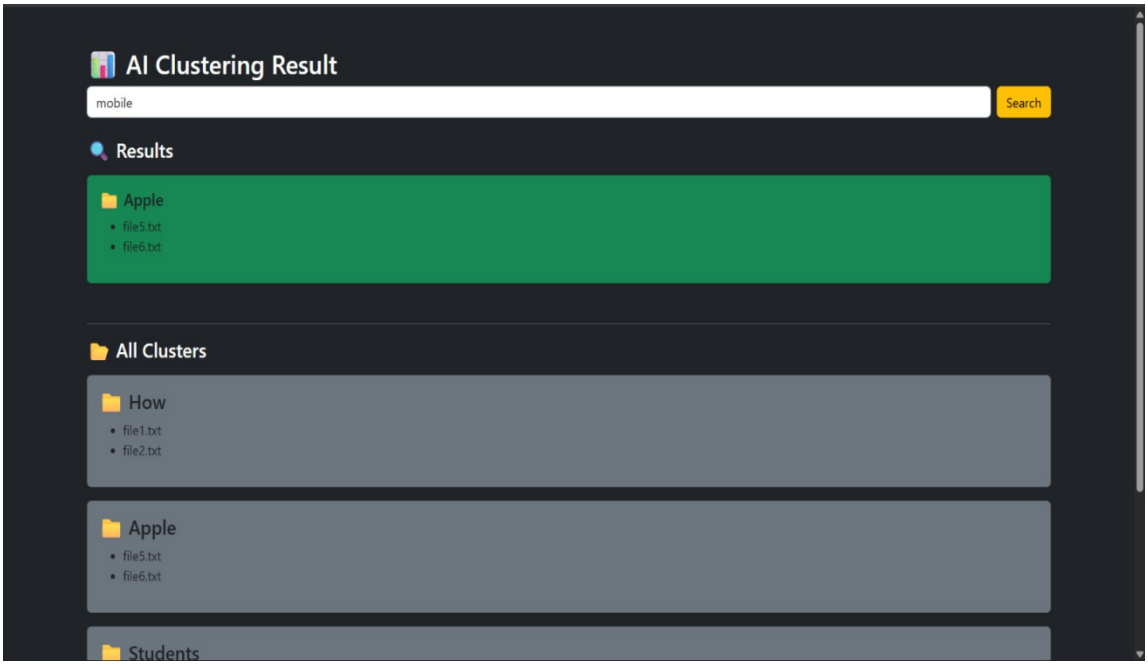


Fig. 5.3.3: Search result for mobile highlighting the matched cluster.

This interface shows the semantic search feature included in the clustering system. A user can input a keyword or query, and the relevant cluster will be retrieved based on the search. The relevant cluster is highlighted to easily locate it. Other clusters remain visible at the bottom for the user’s reference. This feature improves the user experience because it provides direct access to the required data. It also shows the intelligence behind the search, allowing the user to search within the clusters.

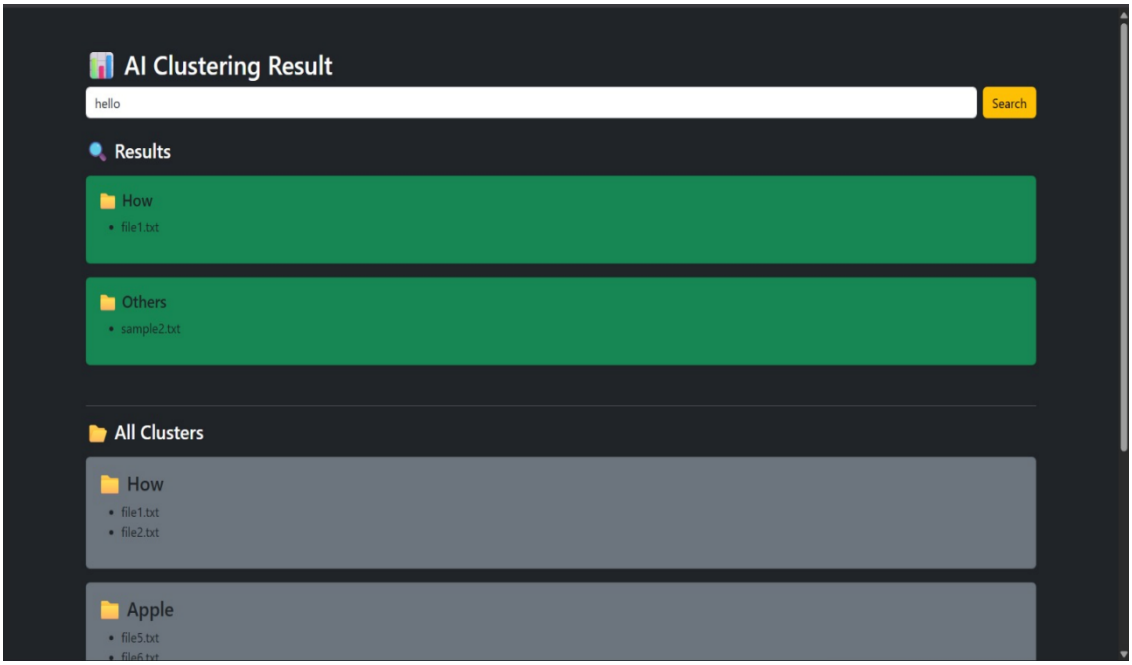


Fig. 5.3.4: Clustered output for the query hello with relevant categories.

The interface for AI Clustering Result acts as a visualization interface for interacting with the clustering results obtained through vector embeddings. The interface contains a dark-themed layout with a search bar in the center, enabling efficient

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

searching. The interface highlights the relevant clusters based on the input provided, which helps in accessibility and interaction with the interface. All the clusters are presented in a systematic manner, which helps in understanding the grouped documents.

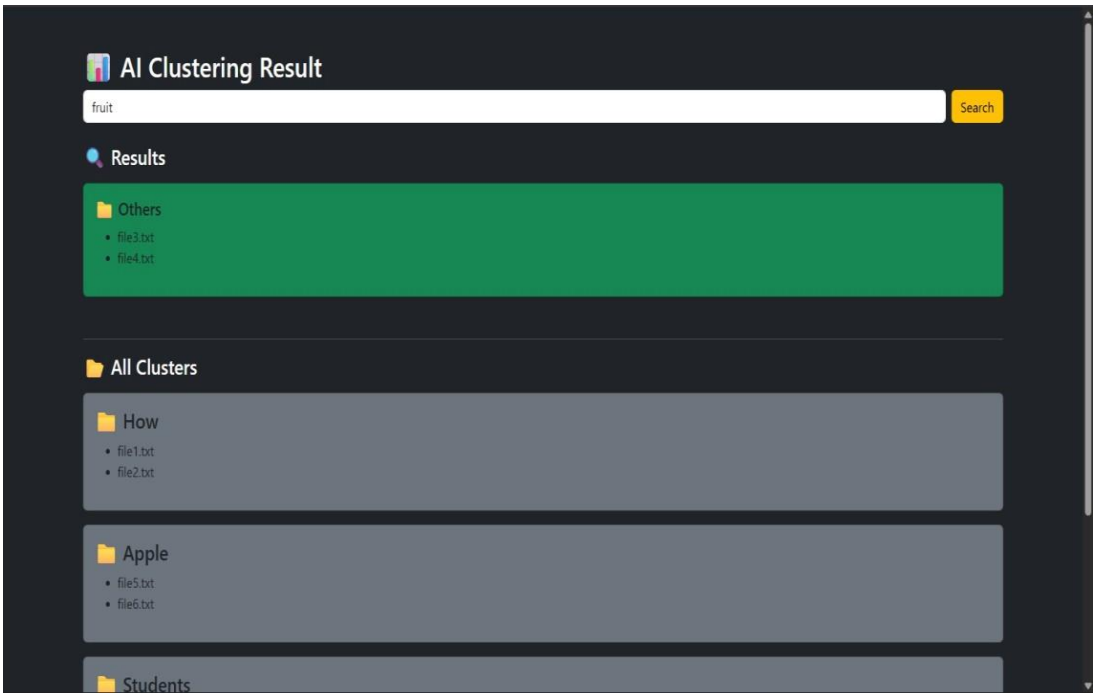


Fig. 5.3.5: Clustering results for the query fruit showing grouped files.

In the interface, the semantic search is clearly demonstrated as the user is prompted to search for a keyword such as "fruit" in the search bar. The system then processes the search result, which is displayed as the most relevant result under the "Others" section rather than the primary clusters. This shows that the search result is not significantly related to the primary clusters but is related in some way. The result is also highlighted to ensure easy identification while still displaying all the clusters.

6. SYSTEM TESTING

6.1 Introduction

System testing is conducted to verify that the complete Semantic Contract Clustering System works correctly according to the specified requirements. In this project, the entire workflow—including contract upload, text preprocessing, BERT embedding generation, clustering, cluster naming, and report generation—was tested. The objective was to ensure accurate semantic clustering of contracts, correct data flow between modules, and proper generation of meaningful cluster names and reports. Additionally, the system was tested to verify that it can efficiently handle multiple contracts simultaneously, maintain stable performance, and produce interpretable results for end users.

6.2 Types of Tests Considered

Unit Testing

Unit testing has been conducted on each module of the system to ensure that each module works correctly on its own. The contract ingestion module has been tested to ensure that the uploaded files (PDF, DOCX, TXT) are correctly read and extracted. The preprocessing module has been tested to ensure that the documents are correctly tokenized, stop words are removed, and lemmatization is correctly applied. The BERT embedding module has been tested to ensure that semantic embeddings are correctly generated for each contract. The clustering module and cluster naming module have been tested independently to ensure that the contracts are correctly grouped and that the clusters are given meaningful names.

Integration Testing

Integration testing was performed to test the communication and data flow between various modules of the system. For

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

this project, communication and data flow between the preprocessing module, BERT embedding module, clustering engine, and reporting module were thoroughly tested. For example, preprocessed contract texts were fed into the BERT embedding module, and then the output was sent to the clustering module. The output from the clustering module was analyzed to check that frequent terms are identified and meaningful names are given to the clusters. This testing was performed to check that all modules are working smoothly and there is proper data flow throughout the system.

System Testing

System testing was performed to test the entire system and check that it meets the requirements specified in the project. In this testing, the entire system is checked from uploading contracts to preprocessing, then BERT embedding, then clustering, and finally generating cluster names and reports. The entire system is checked with varying numbers of contracts to check its performance and stability. During testing, it was observed that the system is able to group contracts with similar clauses even when the clauses are worded differently, and meaningful names are given to clusters.

Functional Testing

Functional testing of the system was done to ensure that all the functional aspects of the system were working as expected. The functionality of uploading various types of contracts into the system was tested for its reliability. The preprocessing module was tested to ensure that the cleaning and tokenization of the text were done correctly. The BERT module was tested to ensure that the embeddings captured the semantic meaning of the input correctly. The clustering module was tested to ensure that the clustering was done correctly, and the cluster naming and report generation module was tested for its clarity and correctness.

Usability Testing

For usability testing, the ease and convenience of interaction with the system were assessed. The usability of the web-based interface was tested to ascertain whether users can upload contracts, view clustering results, and download reports without needing to have technical expertise or make any specific configurations. The usability of the cluster names and the contract reports' organization was also assessed. This type of testing ensures that the system offers users a simple and user-friendly interface to deal with a large number of contracts.

Boundary and Negative Testing

Boundary and negative testing were conducted to ascertain how the system performs in unusual situations. Various situations, including the uploading of corrupted contract files, documents in formats not supported by the system, or excessively large numbers of contracts, were assessed. The system's performance in these unusual situations was observed to ascertain whether it would fail or not. In situations where invalid or unreadable files were uploaded, the system skipped the invalid files and processed the other contracts.

Regression Testing

The regression testing was carried out after changes or modifications had been made to the system to ensure that previously functioning features of the system were not adversely affected. Whenever changes or enhancements, like fixes, were made to certain modules, such as preprocessing, generation of BERT embedding, clustering, or report generation, regression testing was performed to ensure that everything was functioning as it should, including the preprocessing of contracts, generation of embeddings, proper clustering of contracts, and generation of reports.

Black Box Testing

Black box testing was carried out by testing the system from the user’s point of view without taking into account the internal code structure. Sample contracts were uploaded to the system, and the output in the form of cluster assignment and reports was checked. The correctness of the cluster grouping, the interpretability of the cluster names, and the correctness of the reports were checked. Since the internal working of the modules was not taken into account in the testing method, the testing was done purely from the input and output behavior to check if the system was functioning as expected.

White Box Testing

In white box testing, the internal logic and structure of the program code were checked for correctness and efficiency. The code for text preprocessing, BERT embedding creation, clustering, and report writing was checked for correctness and efficiency. Various code paths and conditions were checked for correctness and efficiency, such as handling long contracts, missing values, and different clause structures. The clustering module was checked for the methods implemented for creating meaningful cluster names. This testing was performed to check the internal logic and structure of the system for correctness and efficiency and the absence of major coding errors.

6.3 Various Test Cases Considered

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

ID	Test Case	Descripti on	Test Case Steps	Test Data	Expected Result	Actual Status	Status
TC01	Single Contract Upload	Verify single contract upload	1.Open web interface 2.Upload contract 3.Submit	One valid contract	Contract uploaded and preproce ssing starts	Contract uploaded successf ully. Preproce ssing started.	Pass
TC02	Multiple Contract Upload	Verify multiple contract upload	1.Upload multiple contracts 2.Submit	3–10 contracts	All contracts processe d correctly	All contracts uploaded and processe d successf ully	Pass
TC03	Unsupported File Type	Reject invalid file types	1. Upload .jpg/.exe file	Invalid file	Error message displaye d	Error message displaye d for unsuppor ted file type	Pass
TC04	Empty File Upload	Handle empty file	1.Upload Empty File	Empty file	Error message displaye d	System detected empty file and showed error	Pass
						message	

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

TC05	Preprocessin g Accuracy	Verify text cleaning	1. Upload contract 2. Process	Formatted contract	Text cleaned and tokenize d	Text successf ully cleaned, tokenize d, and processe d	Pass
TC06	BERT Embedding	Verify embeddi ng	1. Pass text to BERT	Preprocess ed text	Embeddi ng generate d	BERT generate d embeddi ngs correctly	Pass
TC07	Similar Clustering	Group similar contracts	1. Upload similar contracts 2. Cluster	Similar contracts	Same cluster assigned	Similar contracts grouped into same cluster	Pass
TC08	Dissimilar Clustering	Separate different contracts	1. Upload different contracts 2. Cluster	Different contracts	Different contracts	Different clusters assigned	Pass

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

TC09	Cluster Naming	Verify cluster labels	1.Run Clustering	Multiple contracts	Meaning ful names generate d	Cluster names generate d based on frequent terms	Pass
TC10	Large Dataset	Test scalabilit y	1.Upload many contracts	1. Upload many contracts	Processe d without crash	System handled dataset without crash	Pass
TC11	Corrupted File	Handle corrupte d input	1. Upload damaged file	Corrupted contract	File skipped with warning	Corrupte d file skipped and warning displaye d	Pass
TC12	Regression Testing	Check after updates	1. Modify code 2. Run system	Sample contracts	System works correctly	System worked correctly after updates	Pass
TC13	Web Interface	Test UI Function ality	1. Upload contracts 2. View results	3–5 contracts	Results displaye d correctly	Clusterin g results displaye d correctly on interface	Pass
TC14	Report	Verify	1. Export	Cluster	Report	Report	Pass

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

	Generation	report output	report	data	generate d correctly	generate d successf ully with cluster details	
TC15	Input Validation	Check filename handling	1. Upload special name file	Special characters	File accepted	File with special character s processe d successf ully	Pass

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

7. CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion

The Semantic-Based Contract Document Clustering System provides an efficient solution for managing and organizing large collections of legal documents. Contracts are often complex and written in varied formats with domain-specific terminology, which makes manual analysis and classification time-consuming and error-prone. By applying Natural Language Processing (NLP) and Machine Learning (ML) techniques, the proposed system addresses these challenges by automatically grouping contracts based on their semantic similarity rather than relying solely on traditional keyword-based methods.

The system analyses the meaning and contextual relationships within contract documents and clusters them into meaningful groups such as employment agreements, lease contracts, and non-disclosure agreements. In addition, the system assigns appropriate names to clusters by analysing frequently occurring terms and clause patterns. This improves interpretability and helps users easily understand the category of documents within each cluster. As a result, the system simplifies document organization, improves retrieval efficiency, and reduces the manual effort required to manage large volumes of contracts.

Another significant advantage of the proposed approach is its ability to handle different document formats such as text and PDF files, making it practical for real- world applications where legal documents are stored in multiple formats. The system enhances accessibility and allows organizations to quickly locate relevant contracts based on their semantic content.

Overall, the proposed system demonstrates how semantic analysis can improve the efficiency of legal document management. By combining NLP and ML techniques, it enables smarter document clustering, better information retrieval, and improved support for legal review processes. The system can serve as a valuable tool for organizations, legal professionals, and document management systems that need to process and organize large numbers of contracts in an accurate and scalable manner.

7.2 Future Enhancement

The proposed Semantic-Based Contract Document Clustering System provides an efficient approach for organizing legal documents based on their semantic meaning. However, several enhancements can further improve its functionality, scalability, and applicability in real-world environments.

One possible future enhancement is extending the system to support additional data formats beyond text and PDF documents. Currently, the system processes textual contract data, but organizations increasingly store information in multimedia formats such as audio recordings, video files, and scanned documents. Integrating speech-to- text and optical character recognition (OCR) technologies would allow the system to extract textual content from audio, video, and image-based documents. This would enable the clustering system to analyse a wider variety of legal materials, making it more versatile and comprehensive.

Another enhancement involves improving the semantic analysis using advanced deep learning models such as transformer-based language models. These models can better understand contextual meaning, legal terminology, and complex clause relationships within contracts. Incorporating such models would increase clustering accuracy and enable more precise identification of contract types and legal obligations.

The system can also be expanded to include intelligent search capabilities. Instead of relying on traditional keyword-based search, the semantic clustering approach could be integrated into search engines to retrieve documents based on meaning and intent. This would allow users to find relevant contracts even when different terminology is used, significantly improving information retrieval efficiency. Additionally, future versions of the system could include automatic clause extraction and risk detection. By identifying critical clauses such as confidentiality, termination conditions, and liability terms, the system could assist legal teams in
quickly reviewing contracts and identifying potential risks.

Finally, implementing cloud-based deployment and scalable architectures would allow the system to process large volumes of documents in real time. This would make the solution suitable for enterprises, legal firms, and regulatory organizations that manage extensive contract repositories.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

8. REFERENCES

8.1 Paper References

[1] M. Steinbach, G. Karypis, V. Kumar. A comparison of document clustering techniques KDD Workshop on Text Mining, 2000.
[2] T. Joachims, Text categorization with support vector machines: Learning with many relevant features European Conference on Machine Learning (ECML), 1998.
[3] W. Xu, X. Liu, Y. Gong Document clustering based on non-negative matrix factorization ACM SIGIR Conference, 2003
[4] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, R. Harshman Indexing by latent semantic analysis Journal of the American Society for Information Science, 1990
[5] A. K. Jain Data clustering: 50 years beyond K-means F. Pedregosa et al. Scikit- learn: Machine learning in Python Journal of Machine Learning Research, 2011.
[6] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[7] D. R. Cutting, J. O. Pedersen, D. Karger, and J. Tukey, "Scatter/Gather: A cluster- based approach to browsing large document collections," in Proc. 15th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval, Copenhagen, Denmark, 1992
[8] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res.,2011.
[9] A. Srivastava, M. Sahami, Text Mining: Classification, Clustering, and Applications CRC Press, 2009
[10]. C. C. Aggarwal, C. Zhai Mining Text Data Springer, 2012

8.2 Websites

[1] IEEE Explore

https://ieeexplore.ieee.org

[2] Research Gate

https://www.researchgate.net/

[3] GeeksForGeeks

https://www.geeksforgeeks.org/

8.3 Text Books

[1] Multilingual Natural Language Processing Applications
[2] Practical Natural Language Processing – Sowmya Vajjala et al.
[3] Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow – Aurélien Géron
[4] Pattern Recognition and Machine Learning

ACKNOWLEDGEMENT

We express our sincere gratitude to our guide **Mr. M. Eswara Rao**, Assistant Professor, in Computer Science and Engineering (Artificial Intelligence and Machine Learning) Department, NNRESGI, who motivated throughout the period of the project and also for his valuable and intellectual suggestions apart from his adequate guidance, constant encouragement right throughout our work.

We would like record to express our profound gratitude to our Project coordinator **Mr. K. Mahesh**, Assistant Professor, in Computer Science and Engineering (Artificial Intelligence and Machine Learning) Department, NNRESGI, for his support and guidance in completing our project and for giving us this opportunity to present the project work.

We profoundly express our sincere thanks to **Dr. G. Sravan Kumar**, Associate Professor & Head of the Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning), NNRESGI, has been of great help in carrying

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

out the project work and is acknowledged with reverential thanks.

We wish to express our sincere thanks to **Dr. G. Janardhana Raju,** Dean School of Engineering, NNRESGI, for providing the facilities for completion of the project.

We wish to express our sincere thanks to **Dr**. **C. V. Krishna Reddy,**

Director, NNRESGI, for providing the facilities for completion of the project.

Finally, we would like to thank Project Review Committee (PRC) members, all the faculty members and supporting staff, Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning), NNRESGI, for extending their help in all circumstances.