Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

Semantic Similarity Analysis of Textual Data

Ali Raza
ali.raza@stud.fra-uas.de

Muhammad Haris Mahmood
muhammad.mahmood@stud.fra-uas.de

Hong Huynh
khon.huynh@stud.fra-uas.de

***Abstract*— This research presents a generalized model for semantic similarity analysis, integrating cosine similarity, dimensionality reduction (PCA and t-SNE), and the Retrieval-Augmented Generation (RAG) approach. The model is structured around the PESTLE framework, making it adaptable to various domains. Through 2D scatter plots and other visualizations, the study demonstrates the effectiveness of semantic similarity analysis across words, PDFs, and datasets. High-dimensional embeddings (1536 dimensions) are generated using OpenAI's embedding model and stored in the Pinecone vector database, where each vector is indexed alongside its corresponding text as metadata. When a query is received, its embedding is computed and matched against stored vectors to retrieve the top K most relevant results. The associated text from these results is extracted and fed into a text generation model, which uses this context to generate accurate and contextually relevant responses. Additionally, the model supports multilingual semantic analysis (mRAG) based on semantic similarity. Using three embedding models (text-embedding-ada-002, text-embedding-3-Small, and Word2Vec) alongside various representation techniques, the results confirm that the proposed approach effectively captures semantic relationships, leading to accurate contextual responses, meaningful visualizations, and improved recommendation quality.**

Keywords — Semantic Similarity, Text Embeddings, Cosine Similarity, Pinecone, OpenAI, mRAG, Dimensionality Reduction, RAG Approach.

I. Introduction

Understanding semantic similarity in textual data is crucial in natural language processing (NLP). It measures how closely the meanings of words, phrases, or documents relate to each other. Various techniques, including cosine similarity, have been used to measure these relationships.

To develop a generalized approach, we incorporate the PESTLE framework, considering Political, Economic, Social, Technological, Legal, and Environmental factors in our semantic similarity tests. This ensures the applicability of our project across multiple domains, rather than being restricted to specific applications like movie recommendations**.** The PESTLE framework allows us to analyze semantic relationships in diverse contexts, making the project versatile and adaptable **[2]**. Our methodology involves comparing words with words, phrases, PDFs, and datasets by computing cosine similarity scores and visualizing the results through 2-D graphs. These plots provide a deeper understanding into real-world contextual relationships between textual data.

A major challenge in visualizing embeddings is their high dimensionality. To address this, we applied dimensionality reduction techniques, including Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE). These methods reduce embeddings to a lower-dimensional space, specifically two dimensions, allowing semantically similar words to cluster together in t-SNE visualizations. PCA retains the most significant variance in the data [11], while t-SNE prioritizes preserving local relationships between data points [3]. In t-SNE, two key parameters—perplexity and theta—significantly influence performance. Perplexity typically ranges from 5 to 50, whereas theta falls between 0 and 1 [3], [7]. Additionally, we employed scatter plots, cosine similarity tables, and output window results to further illustrate these semantic relationships.

Furthermore, we implemented a Retrieval-Augmented Generation (RAG) model that utilizes a vector database (Pinecone) to store embeddings. When a user inputs a query, the system retrieves the top-k relevant embeddings and feeds them into a large language model to generate responses**.** The integration of Pinecone as a vector database ensures efficient storage and retrieval of embeddings, resulting in excellent real-time performance **[4]**. Our approach supports multiple languages, making it applicable for global datasets **[9]**. To validate the system's effectiveness, we test its accuracy by comparing retrieved results with ground truth results. In this study, we employ a dual-validation approach to evaluate the performance of RAG (Retrieval Augmented Generation) approach, combining cosine similarity and ROUGE (Recall-Oriented Understudy for Gisting Evaluation). This combination ensures a RAG accuracy evaluation framework, where cosine similarity captures semantic accuracy and ROUGE validates textual precision. Hence, complementing each other **[1],[8].** Recent studies have used transformer architecture and ensemble techniques to enhance semantic similarity analysis **[5], [6].** These approaches use the attention mechanism to capture long-range dependencies in text, making them highly effective for tasks like semantic similarity and text generation **[5]**.

Currently, 2-D Excel graphs are used to visualize the highest cosine similarity values between genres and datasets, including book descriptions, movie overviews, and music lyrics, highlighting their semantic relationships. The highest cosine similarity indicates that for a particular genre, the corresponding movie, book, or song is the most relevant choice. These graphs provide valuable insights. An end-to-end recommendation system can be developed using the Retrieval-Augmented Generation (RAG) approach. By using RAG, the system can dynamically retrieve relevant content from a database and provide more context-aware and personalized recommendations. This enhancement would make recommendation systems more effective across various domains, including movies, books, and music.**[4].**

The following sections discuss our methodology, implementation, results, and discussion in detail.

II. Methods

This section covers the methodology used in this project, including **text similarity analysis, dimensionality reduction, generalized modeling using PESTLE, and retrieval-augmented generation (RAG).**

A. Semantic and Textual Similarity Evaluation

Semantic similarity is measured using cosine similarity, which determines the closeness of words, phrases, and documents in a high-dimensional embedding space. Word embeddings are generated using pre-trained large language models, and their similarity is calculated using cosine similarity**.** The results are then plotted to visually validate real-world semantic relationships. Cosine similarity is particularly effective for capturing semantic relationships because it focuses on the orientation of vectors rather than their magnitude, making it effective for high-dimensional data **[1]**.

B. PESTLE for Generalized Modeling

To broaden the model beyond a specific domain, we integrate the PESTLE (Political, Economic, Social, Technological, Legal, and Environmental) framework. This ensures that the system can analyze various real-world contexts beyond a particular domain**.** Using diverse datasets from PESTLE-related domains, we demonstrate how cosine similarity can provide meaningful results across multiple sectors. Hence, the PESTLE framework allows the implementation of this project to adapt to different industries, making it a versatile and generalized tool for analyzing semantic similarity **[2]**.

C. Dimensionality Reduction for Visualization

Vector embeddings typically have high dimensionality, with Word2Vec using 300 dimensions, text-embedding-ada-002 and text-embedding-3-Small using 1536 dimensions, and more advanced models reaching up to 3072 dimensions **[12], [13], [14]** . This high-dimensional nature makes direct visualization challenging. To address this, we employ Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce dimensionality to two dimensions. The first principal component is mapped onto the X-axis (scaled: 0–536), while the second component is mapped onto the Y-axis (scaled: -1 to 1), ensuring structured visualization. The scaling is done using the min-max scaling technique. PCA retains the most significant variance in the data **[11]**, while t-SNE focuses on preserving local relationships between data points. Hence, making them effective techniques for presentation **[3]**.
In t-SNE, 'perplexity' and 'theta' are two key parameters that affect how the algorithm works. Perplexity determines the number of neighbors each point considers as a result it helps in balancing local and global patterns in the data. It usually ranges between 5 and 50, and this results in the algorithm staying in stable range **[3]**. On the other hand, Theta controls the speed-accuracy trade-off in the Barnes-Hut approximation. Lower values (0.2 to 0.5) make the results more accurate but slower, whereas higher values (0.5 to 0.8) helps in speeding up the process by summarizing more cells but with a drawback that the precision becomes less **[7]**. Although perplexity results in shaping the visualization and theta impacts how fast the algorithm runs, both the parameters depend on the dataset's size and complexity. To ensure t-SNE produces meaningful and interpretable results we selected the parameters carefully.

D. Retrieval-Augmented Generation (RAG) for Query-Based Analysis

This project implements the Retrieval-Augmented Generation (RAG) approach to retrieve and generate responses to user queries. To set up RAG, we first create embeddings for input text using the OpenAI embedding service and store these embeddings along with their corresponding text as metadata inside the Pinecone vector database. Pinecone is a specialized database designed to store vector embeddings efficiently, enabling fast and accurate similarity searches. Storing the corresponding text as metadata is essential, as it allows retrieval of the actual text during the generation step. This ensures that when a user queries the system, the original context is available for accurate responses.When a user submits a query, the system generates vector embeddings for the input and retrieves the top *k* relevant vectors from Pinecone using cosine similarity.

The response from Pinecone includes these top *k* vectors along with their associated text. This extracted text is then passed to a large language model (LLM), which processes the user query within the given context of the retrieved paragraphs.
Since LLMs are highly capable of understanding and extracting information from context, RAG significantly improves response accuracy. This method allows querying custom datasets, enabling applications to retrieve relevant information from domain-specific data sources before generating responses. When applied to chatbots and similar applications, RAG enhances contextual understanding and user interaction on custom datasets. It is particularly useful in virtual assistants, customer support systems, and any scenario where accurate, context-aware answers are crucial. **[4]**.

E. Multilingual model (mRAG)

In addition to retrieving and generating responses, the RAG can also work with datasets in **multiple languages** by using multilingual embedding models such as OpenAI's **text-embedding-3-small**. This model generates vector embeddings that capture semantic meanings across different languages, enabling cross-lingual similarity search. When a user inputs a query, the system retrieves the top-k semantically similar results, regardless of the language, before passing them to the large language model (LLM) for response generation. To ensure the final response is in the same language as the query, we design the prompt accordingly. This multilingual embedding capability allows the chatbot to handle diverse languages efficiently, making it highly applicable for multilingual customer support, cross-language searches, and globally accessible dictionaries **[9]**.

F. Recommendation using 2-D graphs

In this study, Excel-based 2-D graphs were used to visualize the semantic similarity between genres and datasets, including book descriptions, movie overviews, and music lyrics. The graphs show how user queries (e.g., 'action movies,' 'romantic book,' or 'hip hop songs') match with embeddings of stored data. Cosine similarity was used to measure the relationship between words and datasets, identifying the highest similarity values for each genre. This method helps determine the most relevant movie, book, or song for a given genre based on semantic similarity. While this approach provides an initial analysis of semantic relationships, a more advanced recommendation system can be developed using the Retrieval-Augmented Generation (RAG) approach in future work making it a valuable tool for industries like e-commerce and entertainment [4].

G. RAG Accuracy Calculator

We have designed a RAG Accuracy calculator with the sole purpose of checking whether our RAG model is working accurately or not. The primary metric used is **cosine similarity**, which measures the semantic closeness of the generated text and ground truth text. To complement cosine similarity, we used ROUGE (Recall-Oriented Understudy for Gisting Evaluation) for evaluating textual overlap between generated and ground truth texts. In cases where cosine similarity is unexpectedly low despite the fact that the generated answer is correct, **ROUGE-1 and ROUGE-2** scores provide additional validation of textual similarity. These ROUGE scores assess word or pairs overlap. ROUGE calculates precision, recall, and F1 scores by comparing n-grams (e.g., unigrams for ROUGE-1 and bigrams for ROUGE-2). **Precision** measures the proportion of overlapping n-grams in the generated answer relative to the ground truth, while **recall** measures the proportion of overlapping n-grams in the ground truth relative to the generated answer. The **F1 score**, the harmonic mean of precision and recall, provides a balanced evaluation of textual accuracy. For this project, the ROUGE scores are important when cosine similarity is unexpectedly low, ensuring a more reliable testing of response accuracy. This ensures that even if cosine similarity scores are low when they should not be, ROUGE can validate the textual accuracy of the generated content. Hence, our dual framework design helps us in evaluating both semantic and textual coherence in text generation tasks **[8]**.

H. Semantic Analysis: Visualizations and RAG Testing

Results are visualized using various graphs, including bar graphs, scatter plots, and heatmaps. We have used .NET and Python integration to generate scatter plots, Excel for bar graphs, and Python scripts for heatmaps. The heatmaps were created by generating text embeddings and plotting their vector representations to analyze similarities. These visualizations include word-word, word-phrase, word-PDF, and word-dataset comparisons, helping validate semantic similarity in real-world scenarios. Additionally, cosine similarity tables quantify relationships between phrases on. Scatter plots, created using t-SNE and PCA **[3]**, provide insights into dimensionality reduction. Furthermore, we tested the Retrieval-Augmented Generation (RAG) model by capturing output windows and analyzing its accuracy. These visual representations make it easier for both technical and non-technical audiences to understand the relationships within the data.

III. Implementation

The implementation of this project involves several key components that work together to analyze semantic similarity using various techniques such as including tokenization, word embeddings, similarity computation, dimensionality reduction, RAG pipeline and visualization etc. Each feature of the project is designed in a way to process input text, compute similarities, and present the results meaningfully. Below is a breakdown of the implementation flow:

A. Tokenization and Preprocessing

The first step in processing the input text is tokenization, which involves breaking the text into smaller components such as words, phrases, or paragraphs. This is essential for the next coming steps as it standardizes the text format.

```
"TextHelper.cs"
public List<string> ExtractTextChunks(string pdfFilePath, ChunkType
chunkType = ChunkType.None)
{

}
```

The ChunkType enum specifies the method of text extraction from PDFs: ChunkType. None returning the entire page as a single string, ChunkType. Paragraph splitting text into paragraphs, and ChunkType .Sentence splitting text into individual sentences. ChunkType.None is selected as the default argument because models such as text-embedding-ada-002 and small-3 inherently handle tokenization, preserving the contextual integrity of the extracted text. Selecting Paragraph or Sentence chunking is particularly useful for avoiding exceeding the token limit and to ensure the text is processed effectively.

B. Word Embeddings with ada-002, 3-small, word2vec

This project utilizes three embedding models: OpenAI's text-embedding-ada-002 and text-embedding-3-small, along with Google's Word2Vec embedding model **[10]**. Below is an example for word2vec embedding generation model. After tokenization, the words are transformed into vector representations using the **Word2Vec model**. Hence, resulting in numerical representation of words in a 300-dimensional space.

Word2Vec embedding model:

```
"Word2VecService.cs"
public class Word2VecService
{
    private readonly Dictionary<string, float[]> _wordVectors = new Dictionary<string, float[]>();

    public Word2VecService(string filePath)
    {
      LoadWord2Vec(filePath);
    }

    private void LoadWord2Vec(string filePath)
    {
      foreach (var line in File.ReadLines(filePath))
      {
        var parts = line.Split(' ');
        if (parts.Length < 2) continue;

        string word = parts[0];
        float[] vector = parts.Skip(1).Select(float.Parse).ToArray();
        _wordVectors[word] = vector;
      }
    }

    public float[] GetVector(string word)
    {
      return _wordVectors.TryGetValue(word, out var vector) ? vector : null;
    }
}
```

This service loads pre-trained Word2Vec embeddings from a file and retrieves vector representations for words which are later used in semantic similarity analysis.

OpenAi embedding model:

The given code below generates embeddings from a list of input strings using OpenAI's embedding API. It first calls the _embeddingClient.GenerateEmbeddingsAsync method to get the embeddings for the provided text inputs. Then, it processes each embedding by extracting its numerical vector, converting it into a list, and associating it with the corresponding input text. These processed embeddings are stored in a list and returned. If an error occurs during this process, the method throws an exception with a message indicating that the embedding generation failed.

```
"OpenAIEmbeddingService.cs"
public async Task<List<Embedding>> CreateEmbeddingsAsync(List<string> inputs)
{
   try
   {
      OpenAIEmbeddingCollection collection = await _embeddingClient.GenerateEmbeddingsAsync(inputs);
      var embeddingsList = new List<Embedding>();
      foreach (OpenAIEmbedding embedding in collection)
      {
         ReadOnlyMemory<float> vector = embedding.ToFloats();
         var vectorList = vector.Span.ToArray().ToList();
         var text = inputs[embedding.Index];

         var newEmbeddingModel = new EmbeddingModel(embedding.Index, text, vectorList);
         embeddingsList.Add(newEmbeddingModel);
      }
      return embeddingsList;
   }
   catch (Exception ex)
   {
      throw new InvalidOperationException("Failed to generate embeddings", ex);
   }
}
```

These embeddings help in capturing the meaning of words and phrases in a mathematical form. They are later used in semantic similarity analysis, allowing the system to compare and understand semantic relationships between different pieces of text.

C. Dimensionality Reduction

High-dimensional word embeddings make direct visualization difficult. To overcome this, **t-SNE** (t-Distributed Stochastic Neighbor Embedding) and **PCA** (Principal Component Analysis) are used for dimensionality reduction. In both techniques the vector dimensions are reduced from 1536 to two which can be easily plotted as a scatter plot for analysis.

t-SNE (Dimensionality Reduction Method):

```
"DimensionalityReductionService.cs"
public Matrix<double> ReduceDimensionsUsingTsne(List<List<float>> data, int
targetDimensions = 2)
{
    Console.WriteLine("Performing Tsne");
    var tsne = new TSNE()
    {
       Perplexity = 30,
       Theta = 0.5,
       NumberOfOutputs = targetDimensions
    };
    var dataArray = ConvertListToJaggedArray(data);
    var reducedData = new double[dataArray.Length][];
    for (var i = 0; i < dataArray.Length; i++)
    {
      reducedData[i] = new double[targetDimensions];
    }
    tsne.Transform(dataArray, reducedData);
    return ConvertJaggedArrayToMatrix(reducedData);
}
```

This method applies **t-SNE** to reduce the word embeddings' dimensionality for visualization purposes and at the same time preserving local relationships between data points.

PCA (Dimensionality Reduction Method):

```
"DimensionalityReductionService.cs"
public Matrix<double> PerformPca(List<List<float>> data)
{
    var matrixData = ConvertListToMatrix(data);
    var centeredData = CenterData(matrixData);
    var covarianceMatrix = ComputeCovarianceMatrix(centeredData);
    var eigenDecomposition = covarianceMatrix.Evd();
    var topEigenVectors = eigenDecomposition.EigenVectors.SubMatrix(0,   covarianceMatrix.RowCount, 0, _components);
    var projectedData = centeredData * topEigenVectors;
    return projectedData;
}
```

PCA projects data onto its principal components to retain important variance information while reducing dimensions as a result making the data easier to visualize.

D. Similarity Computation

Cosine Similarity measures how close two vectors are in a multi-dimensional space. In our case, it helps to figure out the semantic similarity between texts by comparing their vector embeddings. A higher cosine similarity score indicates that the corresponding texts are more semantically related.

```
"CosineSimilarity.cs"
public double ComputeCosineSimilarity(List<float> vectorA, List<float> vectorB)
{
    var dotProduct = 0.0;
    var magnitudeA = 0.0;
    var magnitudeB = 0.0;

    for (var i = 0; i < vectorA.Count; i++)
    {
        dotProduct += vectorA[i] * vectorB[i];
        magnitudeA += Math.Pow(vectorA[i], 2);
        magnitudeB += Math.Pow(vectorB[i], 2);
    }
    return dotProduct / (Math.Sqrt(magnitudeA) * Math.Sqrt(magnitudeB));
}
```

This function calculates cosine similarity between two word embeddings. Hence, we can analyze semantic similarity based on the orientation of their vectors.

E. Retrieval-Augmented Generation (RAG) Pipeline

We have designed a Retrieval-Augmented Generation (RAG) pipeline that retrieves and generates responses to user queries. Precomputed embeddings are stored in Pinecone, a vector database optimized for similarity search. When a user submits a query, its embedding is generated and matched against stored embeddings using cosine similarity. The top-k most relevant embeddings and their associated text are retrieved and passed to a large language model (LLM), which generates a context-aware response. This approach enhances response accuracy, making it highly effective for querying custom datasets in chatbot applications and other AI-driven systems. The **Figure 1 [15]** below, explains the RAG pipeline architecture diagram which explains the flow of RAG approach.

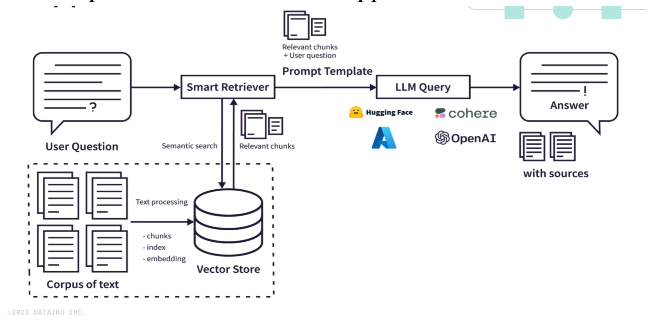


Figure. 1: RAG architecture diagram

```
"ProcessorAI.cs"
{
    string namespaceName = "profiles";
    string indexName = "dr-dobric-index";
    await pineconeSetupService.RunAsync(inputs, indexName, namespaceName); //generate embeddings using openai & upsert
    string query = "Who is dr dobric?";
    var pineconeTopKparagraphs = await pineconeService.QueryEmbeddingsAsync(query, indexName, namespaceName, 3);
    var answer = await textGenerationService.GenerateTextAsync(query, pineconeTopKparagraphs);
}
```

F. RAG-based Chatbot

We implement a chatbot based on this RAG approach, utilizing Pinecone for efficient retrieval and an LLM for generating responses. When a user interacts with the chatbot, their query is converted into embeddings using OpenAI embedding model and matched against stored vectors to retrieve the most relevant context. The retrieved text is then passed to the LLM, ensuring responses are accurate and contextually relevant. This enhances chatbot interactions, making them more informative and capable of handling domain-specific queries effectively.

```
""ProcessorAI.cs"
{
    string namespaceName = "profiles";
    string indexName = "dr-dobric-index";
    await chatbotService.StartChatAsync(indexName, namespaceName);
}
```

This chatbot works fine if the index in pinecone is already created and the vector embeddings are upserted beforehand. Otherwise, the following code can be used to create indexes and upsert the embeddings.

```
"ProcessorAI.cs"
{
    string namespaceName = "profiles";
    string indexName = "dr-dobric-index";
    await pineconeSetupService.RunAsync(inputs, indexName, namespaceName); //generate embeddings using openai & upsert
}
```

G. Multilingual Approach (mRAG)

The RAG approach implemented in this project supports multilingual datasets, enabling retrieval across multiple languages. When a user submits a query, the system retrieves the top-k most relevant results from the vector database, which stores embeddings from a multilingual dataset. The prompt used in the generation step ensures that the language model responds in the same language as the query, maintaining linguistic consistency and accuracy in the generated output.

```
"ProcessorAI.cs"
{
    string namespaceName = "manuals-namespace";
    string indexName = "manuals-index";
    string query = "How to Pair and use wireless headphones?";
    var pineconeTopKparagraphs = await pineconeService.QueryEmbeddingsAsync(query, indexName, namespaceName, 3);
    var answer = await textGenerationService.GenerateTextAsync(query, pineconeTopKparagraphs);
    Console.WriteLine($"\nAnswer: {answer}");
}
```

H. RAG Accuracy Calculator

The given code below evaluates the accuracy of a Retrieval-Augmented Generation **(RAG)** system by comparing generated responses to predefined ground truth answers.

```
"ProcessorAI.cs"
string namespaceName = "profiles";
string indexName = "dr-dobric-index";

List<string> inputQueries = new() {"Where does Dr. Dobric live?"};

List<string> groundTruthAnswers = new() {"Dr. Dobric lives in Frankfurt Rhine-Main Metropolitan Area"};

List<string> generatedResponses = await  ragPipeline.BatchRetrieveAndGenerateResponsesAsync(inputQueries,
indexName, namespaceName, 3);

for (int i = 0; i < generatedResponses.Count; i++)
{
        RagEvaluationResult result = await ragPipeline.EvaluateAccuracy(generatedResponses[i],
groundTruthAnswers[i]);
        Console.WriteLine($"Query: {inputQueries[i]}");
        Console.WriteLine($"Generated Answer: {generatedResponses[i]}");
        Console.WriteLine($"Ground truth Answer: {groundTruthAnswers[i]}");
        Console.WriteLine($"Accuracy Results:");
        Console.WriteLine($"Cosine Similarity: {result.CosineSimilarity}");
        Console.WriteLine($"ROUGE-1 Score: {result.Rouge1Score}");
        Console.WriteLine($"ROUGE-2 Score: {result.Rouge2Score}");
}
```

I. Visualization

With reference to (**Method H**), we utilized various visualization techniques to represent cosine similarity results effectively, providing deeper insights into our implementation's accuracy and its practical applications, such as chatbots and mRAG. We employed scatter plots, heatmaps, and bar graphs to illustrate similarity relationships, making it easier to interpret how different inputs relate semantically. Specifically, scatter plots were generated using both OpenAI and Word2Vec embeddings, employing .NET-Python integration for seamless plotting and visualization. We also utilized Excel to generate bar graphs, providing an alternative representation of similarity scores for clear and structured analysis. This comparative approach helps assess how different embedding models capture semantic nuances. By mapping text embeddings into a lower-dimensional space using techniques like t-SNE and PCA, we visualized clusters of semantically similar data points, validating the quality of embeddings. These visualizations, alongside cosine similarity tables, provided a quantitative and graphical representation of relationships between words, phrases, PDFs, and datasets, making it accessible to both technical and non-technical audiences.

```
"ProcessorAI.cs"
{
   await openAiEmbeddingsDimReductionAndPlotting.RunPipelineAsync(inputs);
   word2VecEmbeddingsDimReductionAndPlotting.RunPipeline(inputs);
}
```

This code runs end-to-end implemented pipelines to generate the vector embeddings (OpenAI, Word2Vec), export the embeddings to CSV files, reduce the dimensions (t-SNE, PCA), create the scatter plots, and save them in the output directory. Additionally, these plots allow us to compare and evaluate different embedding models by highlighting variations in how they represent textual similarities. The integration of .NET and Python ensures a seamless process for generating and visualizing these relationships efficiently.

IV. Results

In this section, we present our findings through various analytical approaches, including heatmaps, dimensionality reduction scatter plots, cosine similarity graphs and tables, accuracy assessments of the RAG chatbot, testing of the RAG pipeline, and evaluation of a multilingual RAG approach.

A. Heatmap:

Heatmaps provide a visual representation of the similarity between different data points by mapping numerical values to colors.

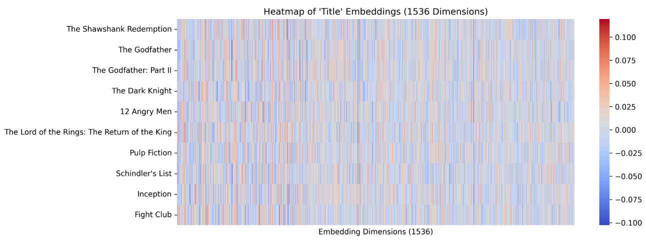


Figure. 2: Movies Embeddings Heatmap

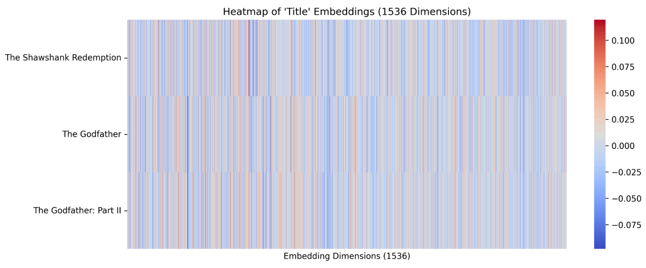


Figure. 3: The Godfather Part I, II

Analysis: In the Figures 2, 3 above, the heatmap compares movie titles based on their embeddings, with similar embeddings appearing in closely matching colors. For instance, movies like *The Godfather* and *The Godfather: Part II* exhibit nearly identical color patterns across all dimensions, indicating that their embeddings are highly similar and share strong semantic relationships.

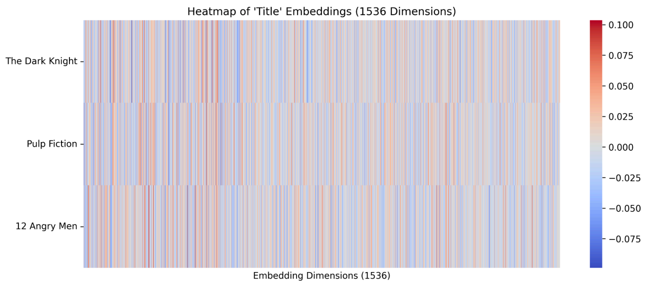


Figure. 4: Pulp Fiction, The Dark Knight

Analysis: Conversely from Figure 4, movies such as *The Dark Knight* and *Pulp Fiction* display some similar patterns but also notable differences in their color distributions, suggesting that their embeddings are not as closely related. While the heatmap offers an intuitive way to identify patterns, its effectiveness is limited. To gain further insights, dimensionality reduction techniques such as PCA or t-SNE can be applied, enabling a more interpretable visualization of relationships in lower-dimensional space.

B. Dimensionality Reduction (Scatter plots):

PCA (Principal Component Analysis) is a dimensionality reduction technique that focuses on capturing the **global variance** in the data **[11]**. Unlike t-SNE, which emphasizes on **local** relationships and often forms clusters (as shown in Figures 5, 7) **[3]**. In PCA data points are spread out along the principal components (Dim1 and Dim2) to explain the maximum variance. This is why the points in the PCA scatter plots (Figures 6, 8) may not form clear clusters but are instead distributed based on their key features.

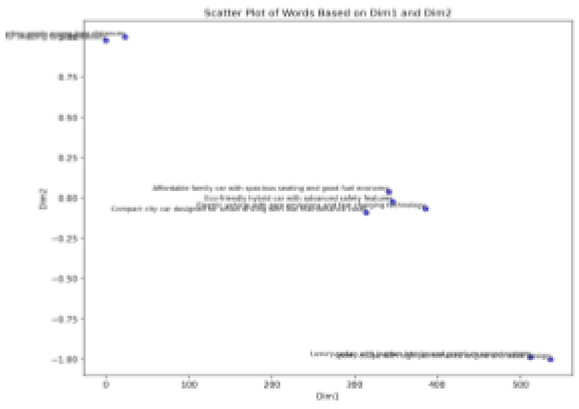


Figure. 5: openai_tsne_scatterplot

Analysis: According to Figure. 5, we plot the two dimensional vectors of input strings after dimensionality reduction using t-SNE. From the figure, it is clear that some of the vectors are grouped together and make a cluster which suggests that these vectors are very close to each other in the vector space. The scatter plot helps visualize how semantically similar or dissimilar the vehicles are based on their features and target markets. For example, luxury vehicles like the **luxury sedan** and **sports coupe** are placed close to each other forming a cluster because they both appeal to consumers who value quality, performance, and premium features. On the other hand, practical and eco-friendly vehicles, such as the **affordable family car, electric vehicle, hybrid car**, and **compact city car**, are grouped together forming a cluster because they focus on affordability, efficiency, and sustainability. Similarly, heavy-duty trucks like the **commercial truck** and **freight truck** form another cluster as they are positioned near each other due to their common purpose of transporting goods over long distances **[3]**.

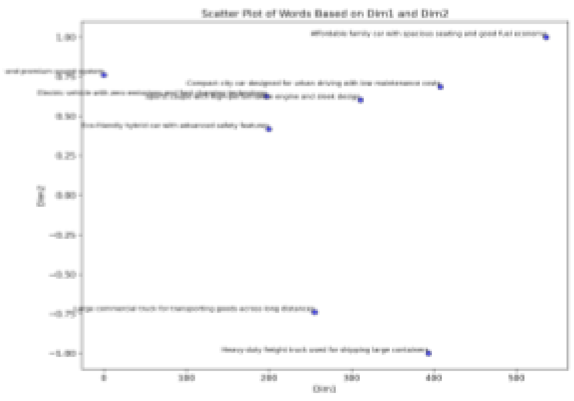


Figure. 6: openai_pca_scatterplot

Analysis: According to Figure. 6, in this **OpenAI PCA** scatter plot, the **Large Commercial Truck** and **Heavy-Duty Freight Truck** are positioned closer but not very close to each other**.** This can be due to the fact that their descriptions emphasize utility and functionality, such as transporting goods across long distances and shipping large containers. This contributes significantly to the variance captured by this dimension. The **Luxury Sedan** is positioned along **Dim2.** This is because its description highlights **premium features** like leather interiors and a premium sound system, which align with the luxury and advanced technology aspects captured by this dimension. However, it was observed in the plot that the **Sports Coupe** was located far away from the Dim 2 even though its variance is similar to the Dim 2. The reason might be that PCA's two dimensions may not have fully retained the semantic relationship or we can say that while reducing dimension to 2-D the dimensions couldn't fully capture the variance **[11]**. The **Affordable Family Car** and **Compact City Car** are positioned in the middle of the **Dim1** and **Dim2 axis.** This is because their descriptions focus on a mix of practicality, affordability, and eco-friendliness, such as spacious seating, good fuel economy, and low maintenance costs. Hence, making them semantically balanced across both dimensions.

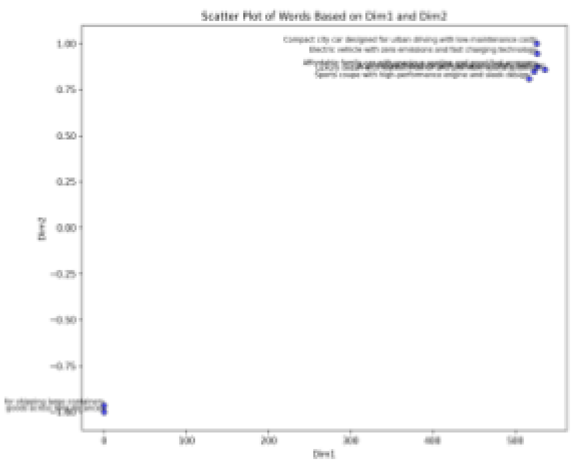


Figure. 7: word2vec_tsne_scatterplot

Analysis: According to Figure. 7, in this **Word2Vec t-SNE** scatter plot, the vehicles are grouped into two distinct clusters based on their semantic similarities. **Cluster 1** includes the Luxury Sedan, Affordable Family Car, Sports Coupe, Electric Vehicle, Eco-Friendly Hybrid Car, and Compact City Car. These vehicles are mainly **small to medium-sized** and are mainly used for **personal or family use**, with features like spacious seating, good fuel economy, low maintenance costs, zero emissions, and premium features. **Cluster 2** consists of the Commercial Truck and Freight Truck, which are **large, heavy-duty vehicles** designed for transporting goods and shipping large containers **[3]**.

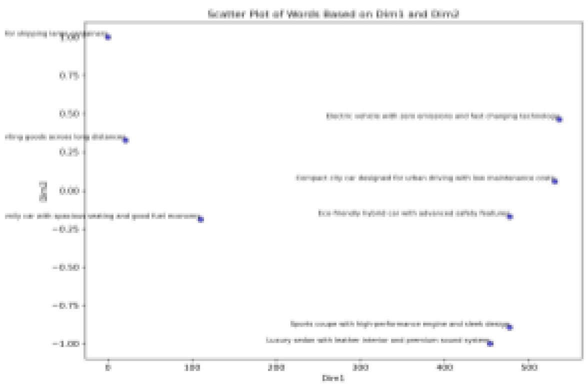


Figure. 8: word2vec_pca_scatterplot

According to Figure 8, in this **Word2Vec pca** scatter plot, the **Sports Coupe** and **Luxury Sedan** are positioned along **Dim1.** This is because their descriptions emphasize performance, luxury, and premium features like high-performance engines, sleek designs, leather interiors etc. as a result contributing significantly to the variance captured by this dimension. The **Electric Vehicle**, **Compact City Car**, and **Eco-Friendly Hybrid Car** are positioned between the **Dim 1** and **Dim 2 axis.** It is because their descriptions highlight a mix of eco-friendliness, practicality, and advanced technology, such as zero emissions, fast charging, low maintenance costs etc. resulting in semantically balanced across both dimensions. The **Commercial Truck** and **Freight Truck** are positioned along **Dim2.** It is because their descriptions focus on utility and functionality, such as transporting goods and shipping large containers. Hence, these descriptions sets them apart and contributes to the variance captured by this dimension.

We have further added two *openai_tsne_scatterplots.* As t-SNE plots clusters therefore, t-SNE scatter plots are used to better demonstrate the idea of semantic similarity.

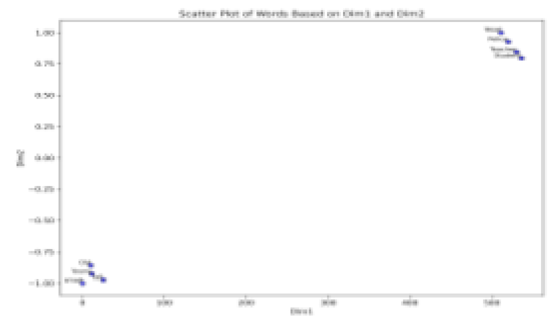


Figure. 9: openai_tsne_scatterplot

Analysis: In Figure 9, we have used words to draw scatter plots. It can be seen that there are 2 clusters formed. Cluster 1 includes teacher, student, police and thief whereas, Cluster 2 includes small, tall, old and young.
As we can observe in Cluster 1, t-SNE has grouped teacher, student, police and thief (which are semantically similar) as they often co-occur. e.g. The teacher teaches the student; The police caught the thief. In these sentences teacher, student and police, thief often come together in sentences. They would definitely not have cosine similarity 1 but have moderately high cosine similarity due to their 'paired roles' and co-occurrence which makes them cluster together. Similarly for Cluster 2, the words small, tall, old, young have high semantics similar to each other as a result forming a cluster. Hence, these clusters portray the true meaning of semantic similarity using t-SNE scatter plots.

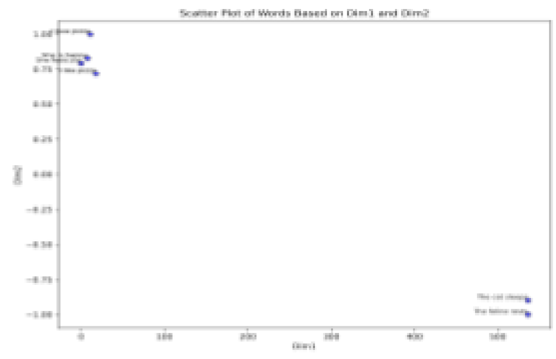


Figure. 10: openai_tsne_scatterplot

Analysis: In Figure 10, we have used sentences to draw scatter plots. It can be observed that there are 2 clusters formed.
Cluster 1: has sentences 'I love pizza';'I like pizza';'She is happy','She feels joy' which are semantically similar to each other due to similar positive sentiments like 'love', 'like','happy','joy'. Cluster 2: has sentences like 'The cat sleeps','The feline rests' which are semantically similar to each other due to similar meaning. Hence, these clusters depict the true meaning of semantic similarity using tSNE scatter plots.

C. Cosine Similarity:

The results of cosine similarities are organized into three main categories: **Words vs Words**, **Words vs PDFs**, **Words vs Datasets**.

It was observed that there were differences in the patterns of the graphs compared for each category below which can be explained by a few key reasons. Firstly, the models used (text-embedding-3-small, text-embedding-ada-002, and Word2Vec) are different in terms of **complexity** and how they understand **relationships** in the data. Secondly, the range of similarity scores [0-0.6] in text-embedding-3-small, [0.6,0.9] in text-embedding-ada-002 and [-0.1,0.6] in Word2Vec affects how the patterns look visually. Apart from that the factors that affect word vectors are the amount and quality of training data, size of vectors and training algorithms. Additionally, the way each model processes and interprets context can lead to variations in where the similarity scores peak or drop. e.g. one model can consider the noun meaning of a word whereas the other would consider the verb meaning. In addition, the word2vec model is a pre-trained model that already has saved embeddings for a large vocabulary of words **[10]**. So with the passage of time words get new meanings or new ways to be represented in sentences. These factors together add up to why the graphs look different, even if the relationships they represent are somewhat similar.

1. Words vs Words

This category focuses on comparing Word-based analyses, specifically using the PESTLE framework and other word lists. The following graphs illustrate the results:

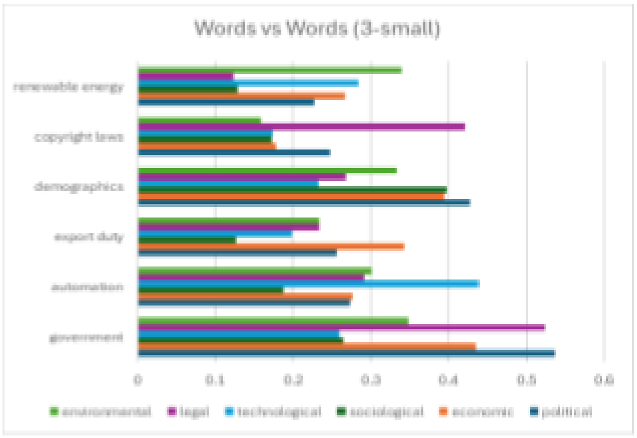


Figure. 11: PESTLE (Words) vs Words (3-small)

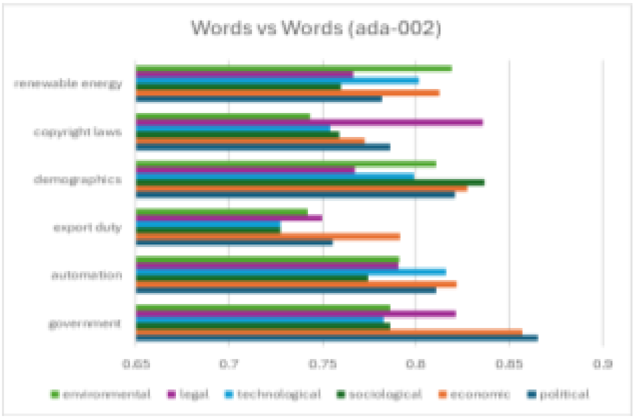


Figure. 12: PESTLE (Words) vs Words (ada-002)

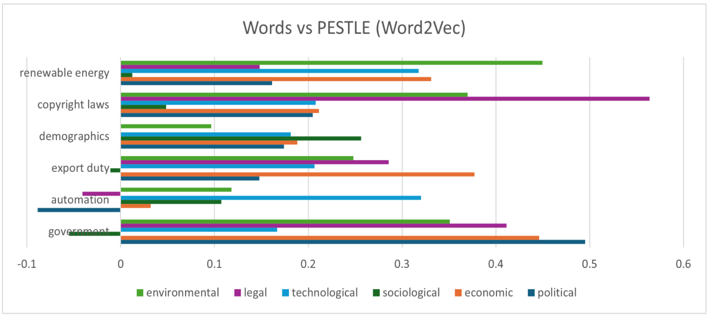


Figure. 13: PESTLE (Words) vs Words (Word2Vec)

The above graphs show the cosine similarities between the PESTLE categories (Political, Economic, Social, Technological, Legal, Environmental) and Words List. The x-axis represents the cosine similarity scores, while the y-axis lists the words from Words List. The PESTLE categories are color-coded for clarity. The embeddings used in the first graph are from 3- small, while the embeddings in the second and third graph are from text-embedding-ada-002 and word2vec, respectively.

Analysis: From Figures 11, 12 and 13 it can be observed that the results (highest cosine similarity) for the three different models match 66%. These include 'environmental' with 'renewable'; 'legal' with 'copyrights'; 'economic' with 'export duty'; 'political' with 'government'. The cosine similarity results differ for 'demographics' and 'automation'. The results of the three models can be different and true at the same moment as it is actually the way with which each model normalizes embeddings that can impact cosine similarity calculations. For instance, 'automation' is semantically similar to 'economic' as with the introduction of automation results in better products and less cost which results in more profits. On the other hand, 'automation' is also semantically similar to 'technological' as automation is the result of the advancement of technology. In another example, it was observed that 'demographics' are semantically similar to 'sociological' as demographic data helps understand social behavior, cultural trends etc of a place, country, region. Moreover, 'demographics' is also semantically similar to 'political' as voting patterns are decided according to demographics like youth population, male vs female percentage; politicians use demographics to design policies for healthcare, education etc. Hence, the results are correct.

2. Words vs PDF

This category focuses on comparing Words with PDFs, specifically using the PESTLE framework in comparison with PDFs. The following graphs demonstrate the results:

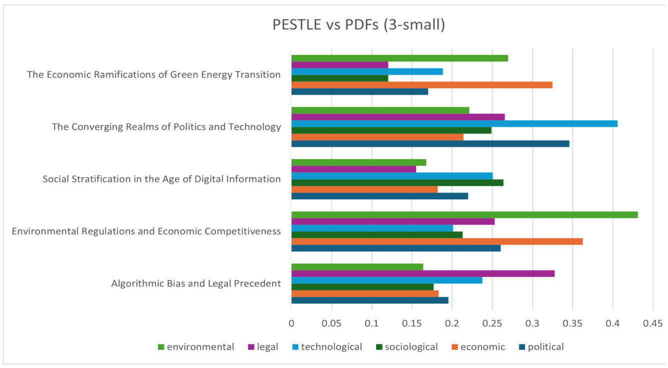


Figure. 14: Words vs PDF's (3-small)

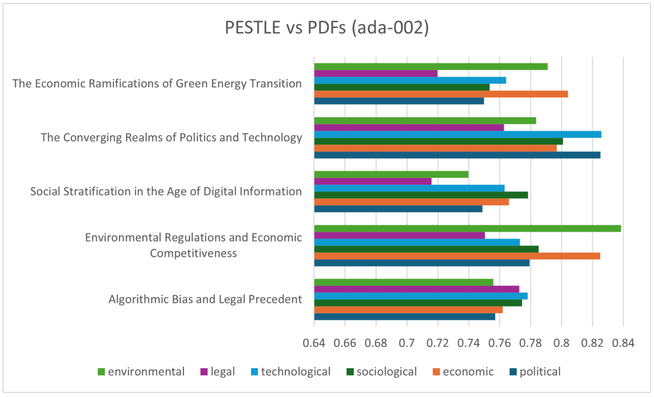


Figure. 15: Words vs PDFs (ada-002)

The above graphs show the cosine similarities between the PESTLE categories (Political, Economic, Social, Technological, Legal, Environmental) and the PDFs. The x-axis represents the cosine similarity scores, while the y-axis lists the PDFs titles. The PESTLE categories are color-coded for clarity. The embeddings used in the first graph are from 3- small, while the embeddings in the second graph are from text-embedding-ada-002.

Analysis: From Figures 14 and 15, it can be observed that the results (highest cosine similarity) for the two different models match 80%. The word 'economic' is rightly semantically similar to 'The Economic Ramifications of Green Energy Technology' as the text talks about how switching to renewable energy affects the economy, including costs, job changes, and financial policies. It explains both the short-term challenges, like job losses, and the long-term benefits, like lower energy costs and economic growth. Since it focuses on money, jobs, and investments, it is strongly related to the word "economic."; similarly, 'environmental' is highest when compared with 'Environmental Regulations and Economic Competitiveness' as the document explores the balance between sustainability and economic policies on environmental regulations. In addition, the role of environmental policies in protecting natural resources, reducing pollution, and addressing climate change as a result impacting businesses, economic competitiveness etc. hence making "environmental" a key theme. It is 'legal' for text-embedding-3-small and 'technological' for text-embedding-ada-002 which have the highest cosine similarity scores when compared with 'Algorithmic Bias and Legal Precedent'. Firstly, discussing its semantic similarity with 'legal', the text discusses how laws struggle to keep up with algorithmic decision-making and bias. It explains legal principles like due process, non-discrimination, and accountability in the context of AI. In addition, it also highlights the need for the introduction of new legal rules to ensure fairness in automated systems. On the other hand, discussing 'technological' semantic similarity, the context of the PDF explains how biases in AI come from flawed data and training methods. In addition, it also discusses the importance of explainable AI (XAI) to make technology more transparent and fair. The results of the two models can be different and true simultaneously, as it's actually the way with which each model normalizes embeddings that can impact cosine similarity calculations.

3. Words vs Dataset

This category focuses on comparing words with datasets, specifically analyzing how words relate to datasets such as book descriptions, movie overviews, and music lyrics. The results of these comparisons are represented in the following graphs. This approach forms the foundation of recommendation systems, which are widely used to suggest relevant items to users based on their preferences. The graphs below display the cosine similarities between the words (genres) and datasets. The x-axis represents the cosine similarity scores, while the y-axis lists the book titles, movie titles, and music names, respectively. The words are color-coded for clarity. For each graph, the embeddings used in the first are from text-embedding-ada-002, while the embeddings in the second graph are from text-embedding-3-small.

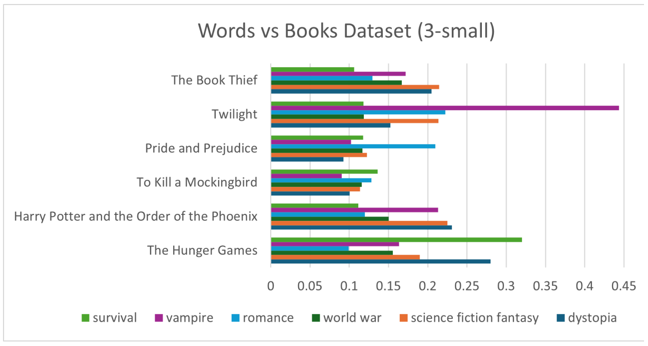


Figure. 16: Words vs Books Dataset (3-small)

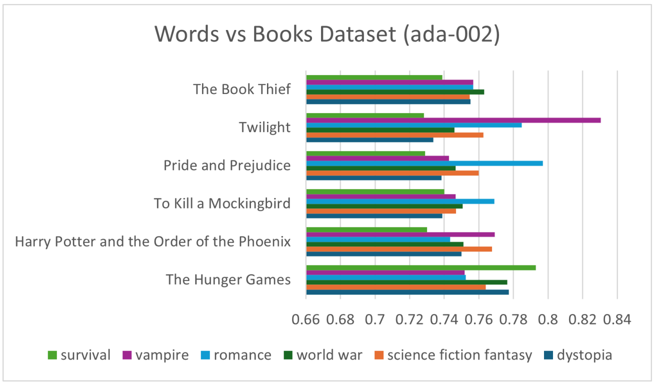


Figure. 17: Words vs Books Dataset (ada-002)

Analysis: From Figures 16 and 17, it can be observed that the results (highest cosine similarity) for the two different models matched 100 %. The graphs would identify 'The Hunger Games' with highest cosine value when someone wants to read a 'survival' genre book. Similarly, it would show 'The Chronicles of Narnia' for 'sorcery'. For 'Romance' lovers, the graphs would highlight 'Pride and Prejudice' and 'Harry Potter and the Order of the Phoenix' for 'sorcery'.

Let's take an example to support the results. The description for the book 'Harry Potter and the Order of Phoenix' is similar to the 'sorcery' genre because it talks about wizards, magic, and supernatural events. It mentions Hogwarts and Defense Against the Dark Arts, which is about learning spells to fight dark magic. The story also includes magical battles, evil forces, and a powerful dark wizard, which all combine semantically as sorcery.

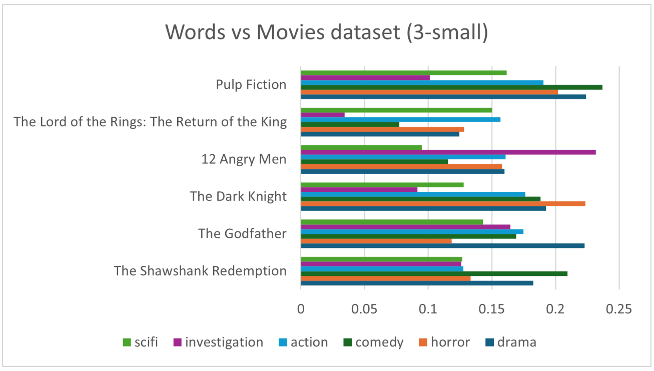


Figure. 18: Words vs Movies Dataset (3-small)

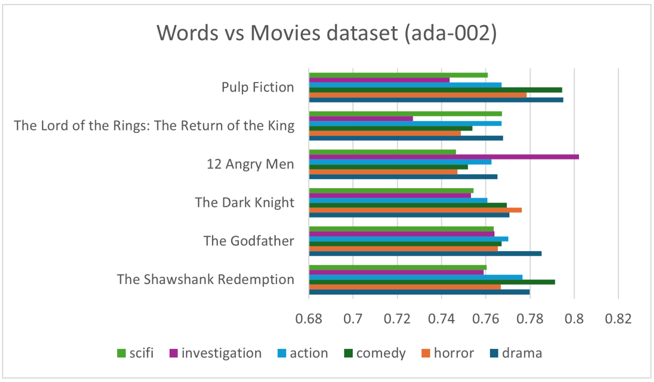


Figure. 19: Words vs Movies Dataset (ada-002)

Analysis: From Figures 18 and 19, it can be observed that the results (highest cosine similarity) for the two different models matched 100 %. The results illustrate that models would suggest '12 Angry Men' for 'investigation'; 'Pulp Fiction' for 'comedy'; 'The Godfather' for 'drama' etc.

For instance, the movie 12 Angry Men is semantically similar to "investigation" because the characters carefully examine the case and debate whether a teenage boy is guilty of killing his abusive father. They challenge each other's perspectives to better understand the case. Similarly, Pulp Fiction is semantically similar to "comedy" due to its use of dark humor and quirky situations. Despite the film's focus on crime and violence, the characters' humor and the unexpected, absurd exchanges lighten the mood, even in serious moments.

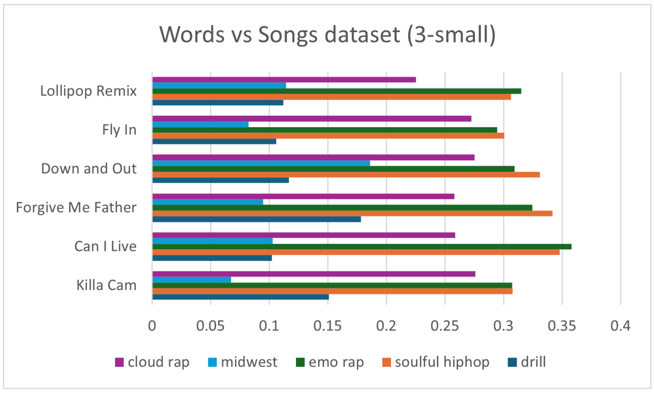


Figure. 20: Words vs Songs Dataset (3-small)

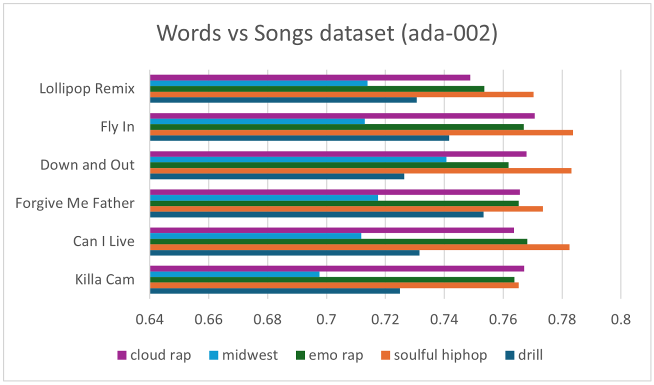


Figure. 21: Words vs Music Dataset (ada-002)

Analysis: From Figures 20 and 21, it can be observed that the results (highest cosine similarity) for the two different models matched 50 %. The graphs would show highest cosine similarity value for 'Fly In'; 'Down and Out'; and 'Forgive me Father' to someone who wants to listen to 'soulful hip hop'. However, the results for the songs 'Lollipop Remix'; 'Can I Live' and 'Killa Cam' are different and true at the same moment as it is the way each model normalizes embeddings that can impact cosine similarity calculations.

For example, the song 'Can I live' has two semantic similarities, such as 'soulful hip hop' and 'emo rap'. For the first similarity, the song is semantically similar to 'soulful hip-hop' because it reflects deep emotions, struggles, and life experiences. The lyrics talk about the hardships of life, dealing with pain, and trying to escape poverty. Coming towards the second similarity, the song is semantically similar to 'emo rap' because it expresses deep personal struggles, pain, and emotional conflicts. The lyrics express stress, crime, survival, and the desire for a better life.

D. Phrase vs Phrase:

This category focuses on comparing Phrases with Phrases, the objective to analyze this test is to examine the semantic similarity between phrases and see if the comparison with the highest cosine value really makes semantical sense. The tests are conducted for both text-embedding-3-small and text-embedding-ada-002.

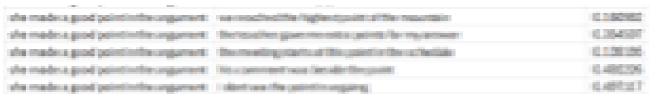


Figure. 22: Phrases vs Phrases (3-small)

Analysis: From Figure 22, it is observed that the maximum cosine similarity value is (0.4971) which is between 'she made a good point in the argument' and 'i don't see the point in arguing'. It is because both are using 'point' to refer to an idea, argument in the discussion or meeting. Therefore, the result is correct as both the sentences have semantic similarity between them as compared to (0.139) which is between 'she made a good point in the argument' and 'the meeting starts at this point in the schedule'. It is because in the first sentence 'point' refers to an idea or statement in a discussion, and in the second sentence 'point' refers to a specific time or location. As a result, they have very low semantic similarity between themselves.

Figure. 23: Phrases vs Phrases (ada-002)

Analysis: From Figure 23, it is observed that the maximum cosine similarity value is (0.859) which is between 'he had to pay a parking fine' and 'the teacher fined him for being late'. It is because 'fine' and 'fined' both refer to some sort of penalty and punishment. Therefore, the high cosine value between these phrases makes sense. On the other hand, the cosine similarity between 'he has to pay a parking fine' and 'the silk is very fine and smooth' is (0.737) which is the lowest in the above-mentioned table. It is justified because in the first sentence 'fine' refers to some sort of penalty and in the second sentence 'fine' refers to delicate, smooth etc. which proves the fact that there is very low semantic similarity between the sentences.

E. Accuracy Assessment of the Chatbot:

The provided snippet (**IMPLEMENTATION F**) demonstrates the initialization of a chatbot service using the RAG (Retrieval-Augmented Generation) approach, where the chatbot retrieves and generates responses based on data stored in **Pinecone**, which is a vector database designed for efficient similarity search and storage. In this case, the profile of 'Dr. Dobric' has been indexed and stored in Pinecone under the index name 'dr-dobric-index' within the namespace 'profiles'. **ChatbotService.StartChatAsync (**indexName, namespaceName**)** was called as a result the chatbot is activated to interact with the user, querying the Pinecone database to retrieve relevant information. The RAG model then processes this retrieved data to generate accurate and contextually appropriate responses as shown in Figure 24.

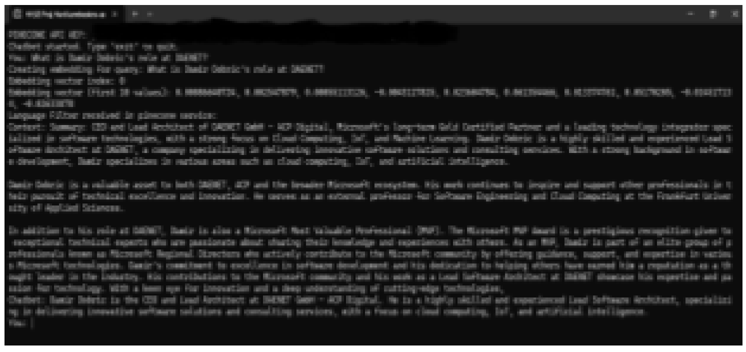
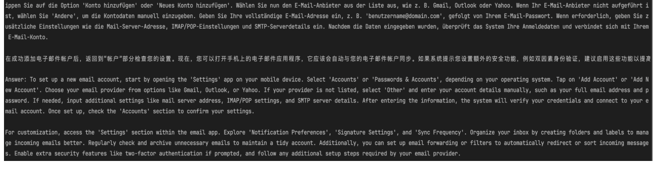


Figure. 24: Testing Chatbot

Analysis: In order to check the accuracy of RAG, we asked the chatbot 10 questions. The criteria to judge its efficiency was to see if it **returns the correct paragraph** (as top1 in top k paragraphs) and it gives the **correct answer** to the questions asked.
The results were remarkable as the paragraph that contained the correct answer was at the first index in the retrieved top k paragraphs. Additionally, the chatbot generated correct and accurate answers for all the queries. As a result, it was observed that the chatbot efficiency was 100% when tested on multiple indexes of pinecone. The screenshots are available here.

G: Multilingual RAG approach:

The provided snippet (**IMPLEMENTATION G**) demonstrates the testing and evaluation of RAG demonstrating significant efficiency in multilingual information retrieval and response generation model by using **semantic similarity** across multilingual contexts. In this test, a query such as ***"How to set up a new email account and customize settings?"*** is processed by retrieving the top k vectors that were generated on a multilingual dataset from the **Pinecone** vector database. **pineconeService.QueryEmbeddingsAsync**(query, indexName, namespaceName, 5) is used for this test.



```
Creating embedding for query: How to set up a new email account and customize settings
Embedding vector index: 0
Embedding vector (first 10 values): 0,027047683, -0,018317826, 0,034965195, 0,015651958, 0,016853316, -0,06517073, 0,04123513, 0,044965055, 0,030503012, 0,058488898
Language Filter received in pinecone service:
Context: For added security and easy management, you can customize your email account settings. Navigate to the 'Settings' section within the email app and explore options such as 'Notification Preferences', '
Signature Settings', and 'Sync Frequency'. You may also want to organize your inbox by creating folders and labels to better manage incoming emails. Regularly check your inbox and archive unnecessary emails to
 keep your account tidy. Lastly, consider setting up email forwarding or filters if you need to redirect or sort incoming messages automatically.

Um ein neues E-Mail-Konto auf Ihrem mobilen Gerät einzurichten, öffnen Sie zunächst die 'Einstellungen'-App auf Ihrem Telefon und wählen Sie 'Konten' oder 'Passwörter & Konten', je nach Ihrem Betriebssystem. T
ippen Sie auf die Option 'Konto hinzufügen' oder 'Neues Konto hinzufügen'. Wählen Sie nun den E-Mail-Anbieter aus der Liste aus, wie z. B. Gmail, Outlook oder Yahoo. Wenn Ihr E-Mail-Anbieter nicht aufgeführt i
st, wählen Sie 'Andere', um die Kontodaten manuell einzugeben. Geben Sie Ihre vollständige E-Mail-Adresse ein, z. B. 'benutzername@domain.com', gefolgt von Ihrem E-Mail-Passwort. Wenn erforderlich, geben Sie z
usätzliche Einstellungen wie die Mail-Server-Adresse, IMAP/POP-Einstellungen und SMTP-Serverdetails ein. Nachdem die Daten eingegeben wurden, überprüft das System Ihre Anmeldedaten und verbindet sich mit Ihrem
 E-Mail-Konto.

在成功添加电子邮件帐户后，返回到"帐户"部分检查您的设置。现在，您可以打开手机上的电子邮件应用程序，它应该会自动与您的电子邮件帐户同步。如果系统提示您设置额外的安全功能，例如双因素身份验证，建议启用这些功能以提高

Answer: To set up a new email account, start by opening the 'Settings' app on your mobile device. Select 'Accounts' or 'Passwords & Accounts', depending on your operating system. Tap on 'Add Account' or 'Add N
ew Account'. Choose your email provider from options like Gmail, Outlook, or Yahoo. If your provider is not listed, select 'Other' and enter your account details manually, such as your full email address and p
assword. If needed, input additional settings like mail server address, IMAP/POP settings, and SMTP server details. After entering the information, the system will verify your credentials and connect to your e
mail account. Once set up, check the 'Accounts' section to confirm your settings.

For customization, access the 'Settings' section within the email app. Explore 'Notification Preferences', 'Signature Settings', and 'Sync Frequency'. Organize your inbox by creating folders and labels to mana
ge incoming emails better. Regularly check and archive unnecessary emails to maintain a tidy account. Additionally, you can set up email forwarding or filters to automatically redirect or sort incoming message
s. Enable extra security features like two-factor authentication if prompted, and follow any additional setup steps required by your email provider.
```

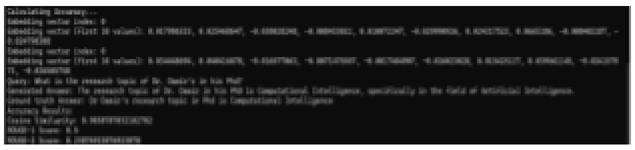
Figure. 25: Multilingual RAG approach

Analysis: From the test in Figure 25, it can be seen that pipecon has returned top 5 results as **context** which are in English, German and Chinese languages. The answer returned from the text generation model is also correct and has the same language as the language of query (as instructed in the prompt passed to the text generation model). The steps to set up an email account on a mobile phone were deliberately listed in different paragraphs in different languages to test the performance of embedding models and text generation models. When queried, pinecone successfully returned the paragraphs that contained the steps to set up an email account, despite the fact that those paragraphs were in different languages. This shows that the embeddings models are smart enough to capture the semantic meanings even if the texts are in different languages and create embeddings that retain the semantic information in the vectors. Secondly, the text generation models are smart enough to translate the text efficiently and produce answers accurately from a given context. The screenshots are available here.

F: Testing RAG Pipeline:

The provided snippet (**IMPLEMENTATION H**) demonstrates the testing and evaluation (RAG accuracy calculator) of a **RAG** pipeline using a set of predefined input queries and their corresponding ground truth answers. The vectors and corresponding texts are stored in the Pinecone vector database. In this case, under the index name ‘dr-dobric-index’ within the namespace ‘profiles’. **BatchRetrieveAndGenerateResponsesAsync** method is used to retrieve and generate responses for a list of input queries, such as questions about Dr. Dobric's location, employer, research topics, awards, and contact details. The generated responses are then compared against the ground truth answers to evaluate their accuracy.

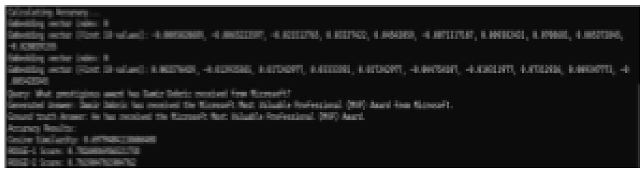
To evaluate the accuracy we use **cosine similarity** to measure the vector-based semantic similarity between the generated and ground truth answers. In addition, we have also applied **ROUGE-1** and **ROUGE-2** metrics to assess similarity at the words (ngram) and phrase (bigram) levels. The main rationale for using ROUGE metrics alongside cosine similarity is to address situations where the generated answers are correct but may not align perfectly in vector space, resulting in lower cosine similarity score. Hence, ROUGE can validate the textual accuracy of the generated content after comparing it with the ground truth text. This dual evaluation approach ensures a better and comprehensive assessment of the RAG, balancing cosine similarity and ROUG metrics. The results, including cosine similarity, ROUGE-1, and ROUGE-2 scores, are shown for each query in Figures 26, 27, illustrating the accuracy and quality of the generated responses.



```
Calculating Accuracy...
Embedding vector index: 0
Embedding vector (first 10 values): [illegible]
Embedding vector index: 0
Embedding vector (first 10 values): [illegible]
Query: What is the research topic of Dr. Damir's in his PhD?
Generated Answer: The research topic of Dr. Damir in his PhD is Computational Intelligence, specifically in the field of Artificial Intelligence.
Ground Truth Answer: Dr Damir's research topic in PhD is Computational Intelligence
Accuracy Results:
Cosine Similarity: 0.903[illegible]
ROUGE-1 Score: 0.5
ROUGE-2 Score: 0.230[illegible]
```

Figure. 26: Testing RAG Pipeline (with high cosine similarity)

Analysis: From the test in Figure 26, it can be seen that the cosine similarity score is high (0.903) which indicates that the embedding vectors of the generated answer and the ground truth answer are very close in the vector space, suggesting strong semantic similarity. In comparison, we can see that the ROUGE-1 (compares word by word) and ROUGE-2 (Pair of words) scores are comparatively low (0.5, 0.230) when comparing the generated answer with the ground truth because the text generation model created an answer that is semantically same as the ground truth but does not match word by word to the ground truth text resulting in lower ROUGE-1 and ROUGE-2 scores.



```
Calculating Accuracy...
Embedding vector index: 0
Embedding vector (first 10 values): [illegible]
Embedding vector index: 0
Embedding vector (first 10 values): [illegible]
Query: What prestigious award has Damir Dobric received from Microsoft?
Generated Answer: Damir Dobric has received the Microsoft Most Valuable Professional (MVP) Award from Microsoft.
Ground Truth Answer: He has received the Microsoft Most Valuable Professional (MVP) Award.
Accuracy Results:
Cosine Similarity: 0.697[illegible]
ROUGE-1 Score: 0.782[illegible]
ROUGE-2 Score: 0.761[illegible]
```

Figure. 27: Testing RAG Pipeline (with low cosine similarity)

Analysis: From the test in Figure 27, it can be seen that the cosine similarity (0.697) is comparatively low even when the generated text and the ground truth text are semantically similar to each other. In this case, the role of ROUGE metric comes into play. Here we can observe that the values of ROUGE are (0.782, 0.761) respectively which are quite **high** and prove the fact that the two answers are similar to each other. Hence, proving the efficiency of the RAG Pipeline Test.The screenshots are available here.

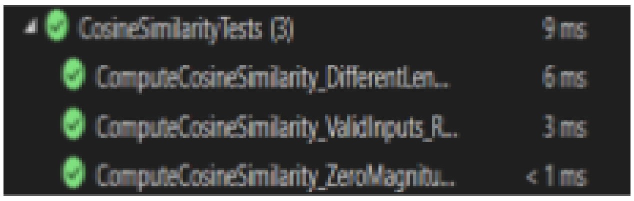
V. UNIT TESTS

Below, we present key unit tests that ensure the accuracy and reliability of our project.

A. CosineSimilarityTests.cs

1. **ComputeCosineSimilarity_ValidInputs:** Validates that the ComputeCosineSimilarity method correctly calculates the cosine similarity between two valid vectors, ensuring the result matches the manually computed expected value within a small delta.
2. **ComputeCosineSimilarity_DifferentLengths**: Confirms that the ComputeCosineSimilarity method returns zero when provided with vectors of different lengths, ensuring proper handling of invalid input.
3. **ComputeCosineSimilarity_ZeroMagnitude**: Verifies that the ComputeCosineSimilarity method throws an ArgumentException when one of the

input vectors has a zero magnitude, ensuring proper error handling.

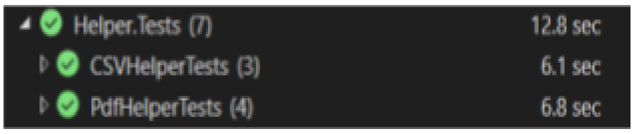


```
◢ ✓ CosineSimilarityTests (3)                           9 ms
    ✓ ComputeCosineSimilarity_DifferentLen...           6 ms
    ✓ ComputeCosineSimilarity_ValidInputs_R...          3 ms
    ✓ ComputeCosineSimilarity_ZeroMagnitu...          < 1 ms
```

Figure. 28: CosineSimilarityTests.cs

B. CsvHelperTests.cs

1. **ExtractRecordsFromCsv_ValidCsv**:
   Ensures that the ExtractRecordsFromCsv method correctly parses a valid CSV file and returns the expected records with accurate field values.
2. **ExtractRecordsFromCsv_FileNotFound**:
   Confirms that the ExtractRecordsFromCsv method throws a FileNotFoundException when provided with a non-existent file path, ensuring proper error handling.
3. **ExtractRecordsFromCsv_MissingCsv**:
   Validates that the ExtractRecordsFromCsv method throws a MissingFieldException when the CSV file is missing required fields, ensuring proper validation of input data.

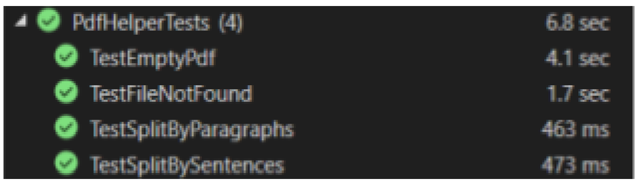


```
◢ ✓ Helper.Tests (7)                    12.8 sec
  ▷ ✓ CSVHelperTests (3)                 6.1 sec
  ▷ ✓ PdfHelperTests (4)                 6.8 sec
```

Figure. 29: HelperTests.cs

C. PdfHelperTests.cs

1. **TestEmptyPdf**:
   Confirms that the ExtractTextChunks method throws an InvalidOperationException when provided with an empty PDF file, ensuring proper handling of invalid input.
2. **TestFileNotFound**:
   Validates that the ExtractTextChunks method throws an InvalidOperationException when provided with a non-existent file path, ensuring proper error handling.
3. **TestSplitByParagraphs**:
   Ensures that the SplitByParagraphs method correctly splits text into paragraphs, validating the expected number of paragraphs.
4. **TestSplitBySentences**:
   Confirms that the SplitBySentences method correctly splits text into sentences, validating the expected number of sentences.

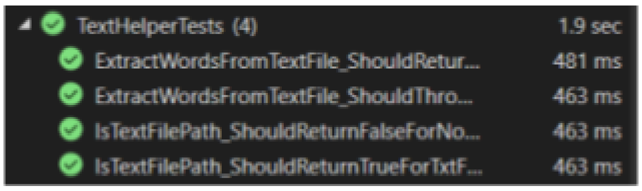


```
◢ ✓ PdfHelperTests (4)                  6.8 sec
    ✓ TestEmptyPdf                      4.1 sec
    ✓ TestFileNotFound                  1.7 sec
    ✓ TestSplitByParagraphs             463 ms
    ✓ TestSplitBySentences              473 ms
```

Figure. 30: PdfHelperTests.cs

D. TextHelperTests.cs

1. **ExtractWordsFromTextFile**:
   Validates that the ExtractWordsFromTextFile method correctly extracts and processes words from a text file, ensuring the expected output.
2. **ExtractWordsFromTextFile_ShouldThrowFileNotFoundException**:
   Confirms that the ExtractWordsFromTextFile method throws a FileNotFoundException when provided with a non-existent file path, ensuring proper error handling.
3. **IsTextFilePath_ShouldReturnTrueForTxtFile**:
   Verifies that the IsTextFilePath method returns true for a valid .txt file extension, ensuring correct file type validation.
4. **IsTextFilePath_ShouldReturnFalseForNonTxtFile**:
   Confirms that the IsTextFilePath method returns false for a non-.txt file path, ensuring correct file type validation.

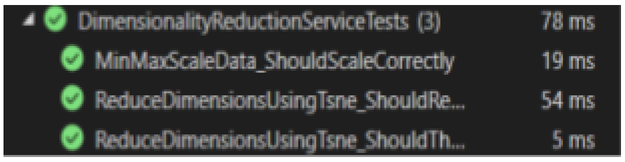


```
◢ ✓ TextHelperTests (4)                               1.9 sec
    ✓ ExtractWordsFromTextFile_ShouldRetur...         481 ms
    ✓ ExtractWordsFromTextFile_ShouldThro...          463 ms
    ✓ IsTextFilePath_ShouldReturnFalseForNo...        463 ms
    ✓ IsTextFilePath_ShouldReturnTrueForTxtF...       463 ms
```

Figure. 31: TextHelperTests.cs

E. DimensionalityReductionServiceTests.cs

1. **ReduceDimensionsUsingTsne_ShouldReduceToTargetDimensions**:
   Validates that the ReduceDimensionsUsingTsne method successfully reduces the dimensionality of input data to the specified target dimensions (e.g., 2 columns), ensuring the output is non-null and correctly shaped.
2. **ReduceDimensionsUsingTsne_ShouldThrowExceptionForInvalidDimensions**:
   Confirms that the ReduceDimensionsUsingTsne method throws an ArgumentException when attempting to reduce data to an invalid target dimension (e.g., 1), ensuring proper validation of input parameters.
3. **MinMaxScaleData_ShouldScaleCorrectly**:
   Verifies that the MinMaxScaleData method correctly scales input data using min-max normalization, ensuring the scaled values match the expected results for both X and Y dimensions.

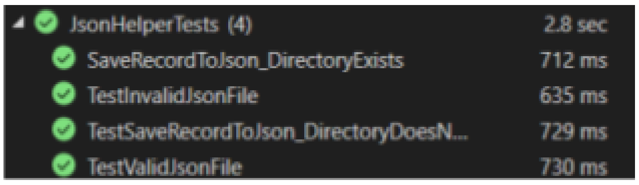


```
DimensionalityReductionServiceTests (3)      78 ms
  MinMaxScaleData_ShouldScaleCorrectly       19 ms
  ReduceDimensionsUsingTsne_ShouldRe...      54 ms
  ReduceDimensionsUsingTsne_ShouldTh...       5 ms
```

Figure. 32: DimensionalityReductionServiceTests.cs

F. JSONHelperTests.cs

1. TestValidJsonFile:
Validates that the GetRecordFromJson method correctly reads and deserializes a valid JSON file into a list of MultiEmbeddingRecord objects. It ensures that the deserialized records match the expected records in terms of attributes and vector data.
2. TestInvalidJsonFile:
Confirms that the GetRecordFromJson method throws an InvalidOperationException with the message "File not found" when provided with a non-existent file path. This ensures proper error handling for invalid file paths.
3.TestSaveRecordToJson_DirectoryDoesNotExist:
Verifies that the SaveRecordToJson method correctly creates a directory if it does not exist and saves the provided list of MultiEmbeddingRecord objects to a JSON file. The test checks that the directory and file are created, and the file content matches the expected records.
4. SaveRecordToJson_DirectoryExists:
Ensures that the SaveRecordToJson method successfully saves the provided list of MultiEmbeddingRecord objects to a JSON file when the target directory already exists. The test confirms that the file is saved correctly and includes cleanup to delete the directory and file after the test.



```
JsonHelperTests (4)                           2.8 sec
  SaveRecordToJson_DirectoryExists            712 ms
  TestInvalidJsonFile                         635 ms
  TestSaveRecordToJson_DirectoryDoesN...      729 ms
  TestValidJsonFile                           730 ms
```

Figure. 33: JsonHelperTests.cs

VI. DISCUSSION

As a first step, tokenization and preprocessing were performed to prepare the data for further analysis. Word embeddings were generated using text-embedding-ada-002, text-embedding-3-small, and Word2Vec models, with cosine similarity used to measure semantic similarity. Dimensionality reduction techniques like PCA and t-SNE were applied to reduce high-dimensional embeddings to 2D for better visualization. This project developed a Retrieval-Augmented Generation (RAG) pipeline, integrating Pinecone for efficient storage and retrieval of data. In addition, it included a RAG-based chatbot for accurate answer retrieval. Furthermore, the model supported a multilingual approach (mRAG), enabling cross-lingual similarity search. The RAG accuracy calculator was designed to evaluate whether the RAG model was functioning accurately. Finally, the results demonstrated that semantic relationships between inputs were clearly shown through various visualizations, and accurate answers were retrieved by the chatbot.

Limitations and Future Directions:

It was observed during the time span of the project that it has some limitations, such as inefficiency in handling reasoning queries e.g. What could be the age of Dr.Dobric if he has been working as a Lead Software Architect since (1998-Present) at DAENET? Future researchers could focus on improving reasoning capabilities by using advanced models. Secondly, on occasional instances, when working with a multilingual dataset, the text generation model unexpectedly produces answers in a language different from the language of the query. Despite implementing a well-structured prompt, this issue persists occasionally. Future researchers can do efficient prompt engineering to fix this issue. Additionally, all the graphs that are plotted using excel or independent python scripts can be done using the .NET-python integration that is already implemented in the project. Due to time limitations, we could not create further scripts inside the project to employ the .NET-python integration for all kinds of plotting. Future researchers should use the .NET-python integration to create additional pipelines for all kinds of plotting. Additionally, due to time constraints, Euclidean distance metrics, 3-D plotting, Network graphs etc. couldn't be tested. Future researchers can implement these techniques to explore semantic similarity in unique representations. In this project, Excel-based 2-D graphs were used to visualize the semantic similarity between genres and datasets. While this approach provides an initial analysis of semantic relationships. Future researchers can make a better recommendation system that can be developed using the Retrieval-Augmented Generation (RAG) approach.

VII. REFERENCES

[1] D. Gunawan, C. A. Sembiring, and M. A. Budiman, "The Implementation of Cosine Similarity to Calculate Text Relevance between Two Documents," *J. Phys. Conf. Ser.*, vol. 978, no. 1, 2018, doi: 10.1088/1742-6596/978/1/012120.

[2] S. P. Vaishali Ugale, "Pestle Based Event Detection and Classification," *Int. J. Res. Eng. Technol.*, vol. 04, no. 05, eISSN: 2319-1163, 2015, doi: 10.15623/ijret.2015.0405112.

[3] L. Van Der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

[4] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Adv. Neural Inf. Process. Syst.*, vol. 2021-April, 2021.

[5] A. Vaswani *et al.*, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 2023-August, 2023.

[6] T. G. Dietterich, "Ensemble methods in machine learning," *Lect. Notes Comput. Sci.*, vol. 1857 LNCS, pp. 1–15, 2000, doi: 10.1007/3-540-45014-9_1.

[7] L. Van Der Maaten, “Accelerating t-SNE using Tree-Based Algorithms,” *J. Mach. Learn. Res.,* vol. 15, pp. 3221–3245, 2014.

[8] C. Y. Lin, “Rouge: A package for automatic evaluation of summaries,” *Proc. Work. text Summ. branches out (WAS 2004)*, 2004.

[9] N. Chirkova et al., “Retrieval-augmented generation in multilingual settings,” no. KnowLLM, pp. 177–188, 2024, [Online]. Available: http://arxiv.org/abs/2407.01463.

[10] word2vec Project, Google Code Archive, 2013. [Online].Available: https://code.google.com/archive/p/word2vec/source, unpublished.

[11] J. Xue, Y. C. Wang, C. Wei, and C. C. Jay Kuo, “Word embedding dimension reduction via weakly-supervised feature selection,” *APSIPA Transactions on Signal and Information Processing, 2024, doi: 10.1561/116.20240046*.

[12][Online].Available: https://www.kaggle.com/datasets/sugataghosh/google-word2vec.

[13][Online].Available: https://openai.com/index/new-and-improved-embedding-model/.

[14][Online].Available: https://platform.openai.com/docs/guides/embeddings#:~:text=By%20default%2C%20the%20length%20of,%2Dembedding%2D3%2Dlarge%20.

[15][Online].Available: https://knowledge.dataiku.com/latest/gen-ai/rag/concept-rag.html.