Semantic Similarity Analysis of Textual Data

Haimanti Biswas   
haimanti.biswas@stud.fra-uas.de

M A B Siddique Naeem

m.naeem@stud.fra-uas.de

Md Abdul Ahad  
 abdul.ahad@stud.fra-uas.de

Muhammad Faraz Abbasi  
m.abbasi@stud.fra-uas.de

Abstract - Semantic similarity analysis is a fundamental task in Natural Language Processing (NLP) that quantifies the degree of similarity between textual data, ranging from search engine to machine translation and document clustering. This project leverages OpenAI’s embedding models – text-embedding-ada-002, text-embedding-3-small, and text-embedding-3-large – to compute cosine similarity scores across different levels of textual data including words, phrases, and documents. The study explores semantic relationships at both word/phrase and document levels, employing well-known entities (e.g., "Angela Merkel" vs. "Government") and contextual document pairs to highlight variations across domains. Implemented in .NET, the project generates similarity metrics, the results are exported to CSV files for further analysis and visualized using a Python-based scatter plots. The study investigates the variation in similarity across different contexts and highlights the importance of embedding-based approaches in capturing semantic relationships. Through well-documented code, rigorous methodology, and intuitive visualization, this work provides a reproducible framework for assessing semantic similarity in diverse text-based applications.

# Introduction

Semantic similarity measures the degree to which two pieces of textual data share meaning. Semantic similarity analysis plays a pivotal role in NLP applications such as information retrieval, recommendation systems, plagiarism detection, question-answering system, and text classification. Traditional approaches to similarity measurement relied on lexical matching techniques, such as Jaccard similarity or term frequency-inverse document frequency (TF-IDF). However, these methods fail to capture the contextual and semantic meaning of words effectively. By quantifying the likeness between textual elements, it enables machines to understand context, infer relationships, and improve decision-making. This project focuses on analyzing semantic similarity using OpenAI's latest embedding models, which convert text into high-dimensional vectors for comparison.

Recent advancements in deep learning have led to the adoption of word embeddings, where words and phrases are transformed into high-dimensional vector representations. Models like Word2Vec, GloVe, and transformer-based architectures such as BERT and OpenAI’s embedding models have significantly improved the accuracy of semantic similarity computations. By leveraging OpenAI’s embeddings and cosine similarity, this project aims to evaluate semantic similarity across words, phrases, and documents in a structured and reproducible manner.

The study addresses two primary objectives:

**1. Word/Phrase-Level Analysis:** Investigating how semantically related terms (e.g., "Angela Merkel" and "Government") compare to unrelated pairs (e.g., "Cristiano Ronaldo" and "Government"). This highlights the role of domain-specific context in similarity metrics.

**2. Document-Level Analysis:** Evaluating similarity between documents on aligned topics (e.g., two articles about machine learning) versus disparate topics to assess the impact of contextual alignment.

The project provides efficient implementation using .NET framework and OpenAI NuGet package, enabling users to input textual data, computes similarity scores using multiple models, exports results to CSV files, and visualizes findings using Python-based scatter plot. The methodology emphasizes reproducibility, leveraging the OpenAI NuGet package and Python scripts for post-processing. By combining theoretical insights with practical implementation, this work contributes to the broader understanding of semantic similarity in NLP.

# Literature Review

Semantic similarity analysis has been a key focus in NLP research for decades.

## Early Approaches to Semantic Similarity

One of the earliest methods for measuring semantic similarity was lexical and syntactic matching, where similarity was determined based on exact word overlap or dictionary-based relationships. WordNet, a lexical database, played a significant role in this era by grouping words into synsets and using path-based similarity measures (Miller, 1995). However, these approaches struggled with issues of polysemy (words with multiple meanings) and synonymy (different words with similar meanings), making them insufficient for complex NLP tasks.

## Statistical and Vector Space Models

With the advancement of computational linguistics, vector space models (VSMs) gained prominence. Term Frequency-Inverse Document Frequency (TF-IDF) was widely used to represent textual data in a high-dimensional space, computing similarity based on term co-occurrence (Salton & McGill, 1983). However, TF-IDF was unable to capture semantic relationships beyond surface-level term matching.

Latent Semantic Analysis (LSA) improved upon VSMs by applying Singular Value Decomposition (SVD) to reduce dimensionality and uncover hidden semantic structures in text corpora (Deerwester et al., 1990). LSA demonstrated effectiveness in capturing meaning beyond word matching but still faced limitations in modeling word order and context.

## Neural Network-Based Embeddings

The introduction of word embeddings marked a major shift in NLP. Word2Vec, introduced by Mikolov et al. (2013), trained neural networks on large corpora to generate dense vector representations of words based on their co-occurrence. Word2Vec's Skip-gram and Continuous Bag-of-Words (CBOW) models captured word meaning more effectively than previous methods.

GloVe (Global Vectors for Word Representation) further enhanced word embeddings by incorporating both global corpus statistics and local context (Pennington, Socher, & Manning, 2014). These models significantly improved semantic similarity computations, enabling better performance in NLP tasks such as document classification and machine translation.

## Transformer-Based Models and Contextual Embeddings

The advent of transformer-based architectures, particularly BERT (Bidirectional Encoder Representations from Transformers), revolutionized NLP. Unlike previous models, BERT produced dynamic embeddings, meaning the representation of a word changed based on its surrounding context (Devlin et al., 2019). This approach significantly improved semantic similarity analysis by considering the entire sentence structure.

More recent advancements include OpenAI's GPT-based embeddings, which leverage self-supervised learning on vast textual data to generate high-quality embeddings for semantic similarity tasks (Brown et al., 2020). The OpenAI embedding models used in this project—text-embedding-ada-002, text-embedding-3-small, and text-embedding-3-large—build on these transformer-based approaches to provide accurate and scalable semantic similarity measurements.

## Current Study Contribution

This project builds upon the existing literature by implementing OpenAI’s embedding models within a structured framework in .NET. By computing cosine similarity between embeddings at word, phrase, and document levels, the study provides insights into the effectiveness of modern NLP embeddings. The project also incorporates visualization techniques to analyze trends in similarity scores, making it a valuable contribution to the field of semantic similarity analysis, aligning with recent trends in explainable AI (Samek et al., 2021).

## Applications and Challenges:

* **Applications:** Semantic similarity is critical for chatbots (e.g., matching user queries to responses), plagiarism detection, and clustering (e.g., grouping news articles by topic).
* **Challenges:** Variability in similarity scores across models (e.g., ada-002 vs. text-embedding-3-large) and the need for domain adaptation remain active research areas.

# Methodology

Our methodology centers on achieving precise reconstruction of the original input data. Initially, various types of input data are provided, including numerical values ranging from 0 to 99 and images. For image data, an Image Binarizer is used to convert the images into binary representations.  
  
Subsequently, the input data undergoes transformation into int[] arrays, each consisting of 0s and 1s. These arrays serve as the sole source of input data for our experiment.  
Next, the Hierarchical Temporal Memory (HTM) Spatial Pooler is employed to generate Sparse Distributed Representations (SDRs) from these encoded int[] arrays.  
  
Following this, the reconstruction process begins. Utilizing the Neocortexapi's Reconstruct() method, the original encoded representations are meticulously reconstructed using the permanence values returned by the Reconstruction method.  
  
Our methodology concludes with two primary visualizations: heatmaps and int[] sequences. Additionally, we perform a similarity check between the original input arrays and the reconstructed input arrays to evaluate the accuracy of the reconstruction process.  
  
The main objective is to evaluate the HTM algorithm's ability to accurately reconstruct inputs using the Neocortexapi's Reconstruction method, focusing on faithfully recreating the structure of the original encoded int[] arrays*.*  (Figure 1)

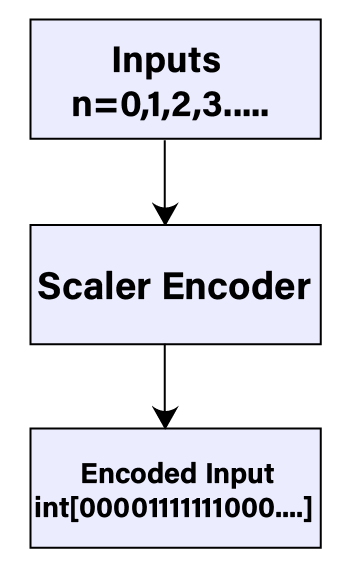
Methodology Flowchart


.

*Figure 1: Graphical Representation of the Experiment*

## Input Types for the Experiment

### **Input for Numerical Values:** For numerical values ranging from 0 to 99, a scaler encoder is utilized to transform each numerical value into an encoded representation consisting of 200 bits. These encoded representations are then transformed into int[] arrays, where each array represents a numerical input. (Figure 2)



*Figure 2: Encode inputs for Numerical Inputs*

### **Input for Images**: For images, an Image Binarizer is used to convert them into binary representations. The size of the resulting encoded arrays matches the dimensions of the original images. For instance, a 28x28 pixel image results in an encoded array of size 784, with each pixel represented by a binary value. (Figure 3)

A diagram of a binary system

Description automatically generated

*Figure 3: Encoded Inputs for Images*

Both types of encoded arrays serve as the input data for our experiment, aiming to reconstruct the original input data with precision using the Hierarchical Temporal Memory (HTM) framework.

## Reconstruction Method

Prior to initiating our project and delving into the complexities of Hierarchical Temporal Memory (HTM) and the Neocortex API, we laid a solid groundwork by watching enlightening HTM School videos on YouTube. These concise yet insightful videos provided a clear overview of crucial elements such as HTM configuration, encoder dynamics, Spatial Pooler functionalities, and more. This preparatory step equipped us with essential insights, ensuring a well-informed approach to our project endeavors. The HTM School videos became a key resource, streamlining our understanding of HTM principles and facilitating navigation through the NeoCortex API functionalities.

An essential part of the NeoCortex API is the Reconstruct method, which is meant to reverse the process and reconstruct the initial input from Sparse Distributed Representations (SDRs) produced by the Spatial Pooler. (Figure 4) An elaborate description of the method's operation is provided below:

The Reconstruction Method Diagram


*Figure 4: Graphical illustration of The workflow of the Reconstruction Method*

### **Input Validation:** The method begins with thorough validation, throwing an ArgumentNullException if the input array of active mini-columns is null. If the input array is not null then it goes further to complete the Reconstruction operation.

### **Column Retrieval:** Retrieve the list of columns associated with the active mini-columns from the connections. The activeMiniColumns parameter in the Reconstruct method is an array of integers that indicates the indices of mini-columns that are considered active. A collection of cortical neurons is called a mini-column. In hierarchical temporal memory (HTM) or related algorithms, input patterns meeting specific criteria, like overlapping patterns or sensory inputs, cause mini-columns to fire. The present technique employs activeMiniColumns to determine the active mini-columns. Subsequently, the dictionary containing the cumulative permanence values of the synapses connected with these active mini-columns is reconstructed. These permanence values are essential for determining the strength of connections between neurons in the cortical region.

### **Reconstruction Process**: Iterate through each column, accessing the synapses in its proximal dendrite.

**Proximal Dendrite:** In HTM, each column in the cortical model contains a proximal dendrite.

* The proximal dendrite receives input from neighboring columns or directly from the input data.
* It serves as the primary input interface for each column, responsible for capturing spatial patterns in the input data.
* Proximal dendrites are associated with synapses, which represent the connections between the input space and the columns in the HTM network.
* The proximal dendrite is crucial for the spatial pooling process, where columns compete to represent spatial patterns in the input data.

**Synapses:** Synapses are the connections between neurons or dendrites in biological brains or between computational units in artificial neural networks.

* In the context of HTM, synapses connect the proximal dendrites of columns to the input data or other columns within the network.
* Each synapse has an associated permanence value, which determines the strength of the connection.
* The permanence value represents the likelihood that a synapse will contribute to the activation of the connected column in response to input.
* Synapses play a critical role in learning and adaptability within the HTM framework, as the permanence values are updated based on input patterns and network activity.

For each synapse, accumulate the permanence values for each input index in the reconstructed input dictionary. Update the reconstructed input dictionary, considering whether the input index already exists or needs to be added as a new key-value pair. The method concludes by returning the reconstructed input as a dictionary, mapping input indices to their associated permanence.

## Proposed Method to Visualize Permanence Values

The goal of this segment is to use the Reconstruct method offered by the Spatial Pooler (sp) to reconstruct permanence values from the active columns acquired during the learning phase of the experiment. The below flow chart illustrates that how we implemented our proposed method to visualize the permanence values. (Figure 5)

A diagram of a process flow

Description automatically generated

*Figure 5: Graphical Representation of Running Reconstruction Method.*

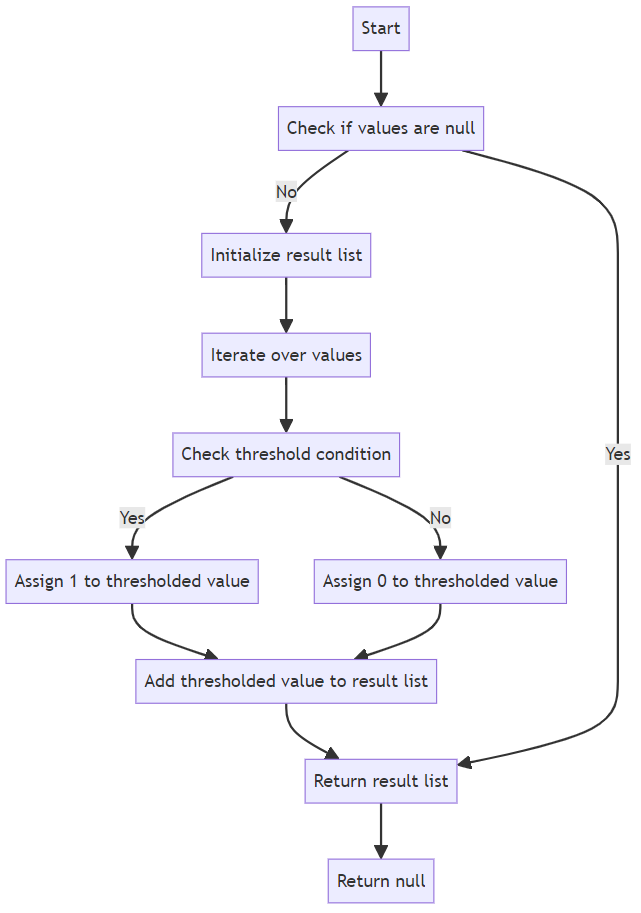
Our proposed method for visualizing permanence values and measuring similarity for numerical inputs initiates by encoding each numerical value using the provided encoder. This encoding process generates a Sparse Distributed Representation (SDR) for each numerical input, capturing its essential features. Subsequently, we compute the active columns in the spatial pooler for the given input SDR, refraining from learning to preserve the original input characteristics. The next crucial step involves reconstructing the permanence values for the active columns, accomplished using the Reconstruct method provided by the spatial pooler. A dictionary called allPermanenceDictionary is created to cover all potential numerical input indices comprehensively. Initially, the permanence values for active columns are included in this dictionary to ensure a thorough representation. To maintain consistency and ease of analysis, the dictionary is organized in ascending order based on keys, ensuring a uniform structure across all permanence dictionaries used. Notably, the reconstructed permanence values obtained from active columns only cover a portion of all possible input indices. To ensure a complete representation of all input bits and to capture the full scope of conceivable input indices, permanence values for inactive columns are also taken into account. These inactive columns, which do not contribute actively to the input representation, are assigned a default permanence value of 0.0. This meticulous process ensures the allPermanenceDictionary comprehensively represents all input indices, enabling in-depth analysis and visualization of the spatial patterns learned by the HTM network. The reconstructed permanence values are then subjected to normalization based on a threshold value, facilitating comparability and further analysis. Following normalization, the permanence values are converted into a list of integers, streamlining subsequent processing steps. To evaluate the fidelity of the reconstruction process, Jaccard similarity is computed between the original encoded inputs and the normalized permanence values, quantifying the resemblance between the two representations. Finally, leveraging the heatmap data and the normalized permanence values, we generate visually informative heatmaps, offering insights into the spatial patterns learned by the HTM network from numerical inputs. Additionally, we create similarity plots, visually depicting the degree of similarity between the original encoded inputs and the reconstructed inputs, facilitating deeper understanding and validation of the reconstruction process.

Similarly, for image inputs, our proposed methodology follows a parallel trajectory, albeit tailored to accommodate the unique characteristics of image data. We commence by reading and binarizing each image file from the designated training folder, a crucial preparatory step in preparing the image data for spatial pooling. Once binarized, spatial pooling is applied to the images, leading to the identification of active columns representing salient features within the images. The subsequent reconstruction of permanence values for the active columns mirrors the process employed for numerical inputs, ensuring a thorough representation of the image space. Employing the allPermanenceDictionary, we meticulously catalog all possible input indices, addressing any gaps to guarantee a comprehensive view of the input domain. As with numerical inputs, normalization and thresholding are applied to the reconstructed permanence values, followed by similarity computation to assess the fidelity of the reconstruction process. Ultimately, the visualization efforts culminate in the generation of heatmaps, providing a visual narrative of the spatial patterns discerned from image inputs. Similarly, similarity plots are crafted to offer a graphical depiction of the resemblance between the original encoded inputs and the reconstructed inputs, fostering deeper insights and validation of the reconstruction process in the context of image data.

# Implementation Overview

Embarking on our implementation journey within the Hierarchical Temporal Memory (HTM) framework, we aim to visualize and understand permanence values for both numerical and image inputs. This part of this paper is focused on some important parts of our Implementation. Our goal is to develop a clear method that not only adjusts these values but also helps us compare and understand them better. Through exploring how to set certain thresholds, creating heatmaps, and checking for similarities, we want to uncover the hidden patterns and connections stored within the HTM network. This work is crucial for understanding how HTM processes information over time and across different types of data. Now, let's dive into the details of how we're going to start by adjusting the permanence values using thresholding probabilities.

## Normalizing Permanence To ensure the consistency and comparability of permanence values obtained from the spatial pooler within the Hierarchical Temporal Memory (HTM) framework, we implemented a thresholding function aimed at normalizing these values. This function plays a crucial role in the visualization and analysis process by transforming continuous permanence values into a binary representation. (Figure 6)

**

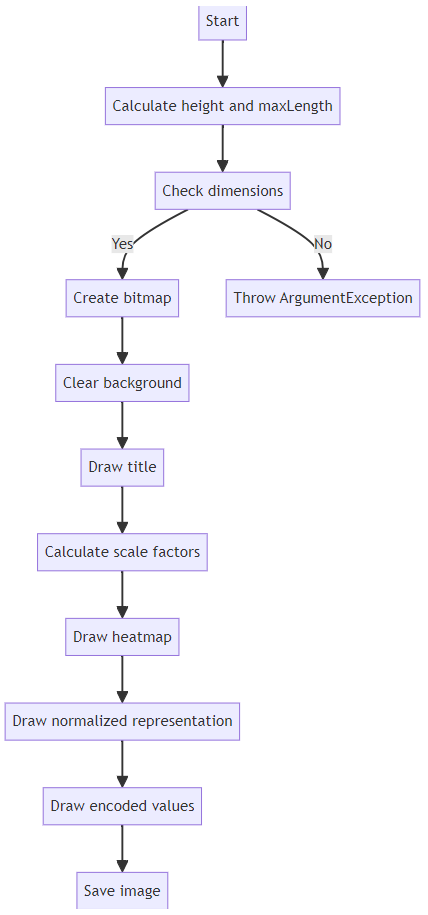
*Figure 6: Graphical Representation of the Implemented Thresholding Probability Function*

The determination of an appropriate threshold value is vital to this normalization process. Through experimentation and iterative debugging, we identified optimal threshold values for both numerical and image inputs. For numerical data, the selected threshold value was empirically set at 8.3, while for image inputs, it was determined to be 30.5. These thresholds were chosen based on extensive experimentation and observation of the output, ensuring that the normalized permanence values maintain fidelity to the original encoded inputs. By systematically varying the threshold values and comparing the resulting outputs with the original encoded inputs, we refined the normalization process to achieve the desired balance between preserving input characteristics and facilitating meaningful analysis. This meticulous approach underscores the importance of parameter selection in effectively visualizing and interpreting permanence values within the HTM framework, ultimately enhancing our understanding of hierarchical temporal processing in both numerical and image data domains.

## Visualization With Combined Heatmap

In the visualization of the heatmap, it's crucial to understand that the color interpolation serves as a visual indicator of the permanence values associated with each input index. The concept of "heat" within the heatmap corresponds directly to the strength of the connection between input data and columns within the Hierarchical Temporal Memory (HTM) network. In this context, higher permanence values are represented by warmer colors, indicating a stronger likelihood that the associated synapse will contribute to the activation of the connected column in response to input. Conversely, cooler colors signify lower permanence values, suggesting a diminished influence of the corresponding synapse on column activation. Essentially, the color gradient in the heatmap provides a visual representation of the network's interpretation of the input data, with hotter colors denoting stronger connections and colder colors indicating weaker or nonexistent connections.

It's important to keep in mind the underlying principles of synapses and permanence values within the HTM framework. Synapses act as connections between the proximal dendrites of columns and the input data or other columns within the network. Each synapse is assigned a permanence value, which serves as a measure of the strength of the connection. This permanence value represents the likelihood that the synapse will contribute to the activation of the connected column in response to input. By visualizing the heatmap and interpreting the color gradient, we gain valuable insights into how these synapses are weighted and how they influence the network's processing of both numerical and image inputs. (Figure 7) We are illustrating the implementation of the Combined Visualization process given below:



*Figure 7: Graphical representation of Draw 1D Heatmap Function*

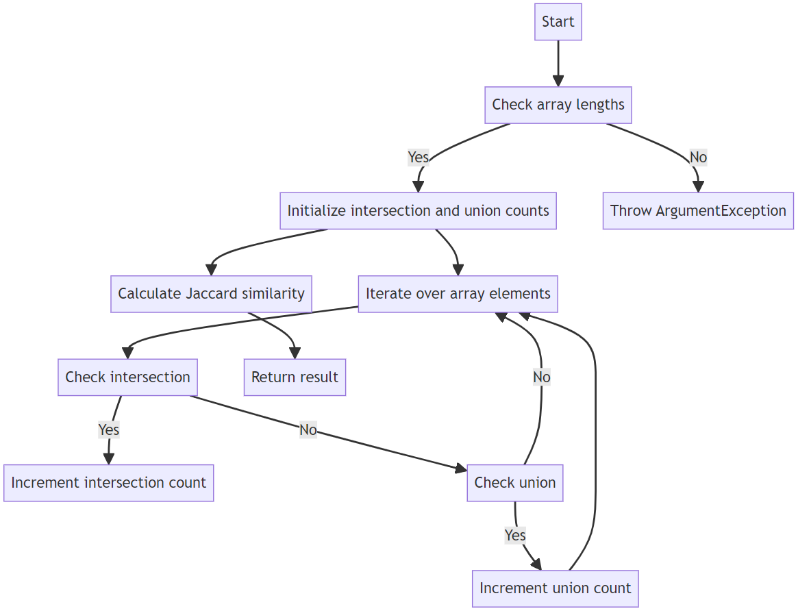
The Draw1dHeatmap function is instrumental in our visualization efforts, as it seamlessly integrates heatmap data with normalized permanence values and original encoded inputs. This integration allows us to create holistic visual representations that accurately depict the intricacies of the network's learning process. Each heatmap is meticulously designed to highlight the intricate relationship between input indices and their corresponding permanence values, offering a visual narrative of the HTM network's interpretation and processing of the input data.

Furthermore, by incorporating the original encoded inputs alongside the heatmap data, we gain valuable insights into how the input data is translated and represented within the HTM network. This comparative analysis enables us to evaluate the accuracy of the network's representation and pinpoint areas for potential refinement and enhancement.

## Similarity Calculation:

On our Experiment to calculate the similarity to validate the output which is the Reconstructed Input we have used the Jaccard similarity coefficient. The Jaccard coefficient, also known as the Tanimoto coefficient, assesses similarity by comparing the intersection of objects to their union. [7]. The Jaccard similarity coefficient is well-suited for measuring the similarity between two sets of binary data, such as the original encoded inputs and the reconstructed inputs in our scenario. Mathematically, the Jaccard similarity coefficient (Equation 2) between two sets is defined as the ratio of the size of the intersection of the sets to the size of their union:

*Equation 2: The Similarity Calculation Using Jaccard Similarity coefficient.*



*Figure 8: Graphical Representation of Implemented Jaccard Similarity Coefficient Calculation Function.*

In this implementation, the method takes two integer arrays as input, representing binary values. It first validates that the arrays have the same length, throwing an exception otherwise. Then, it iterates through the arrays to count the number of intersecting elements (both arrays have a value of 1 at the same index) and the total number of unique elements (either array has a value of 1 at the index). Finally, it computes and returns the Jaccard similarity coefficient by dividing the intersection count by the union count, ensuring a normalized measure between 0 and 1. (Figure 8)

## Similarity Graph Visualization:

Visualizing the similarity calculation graph offers valuable insights into the performance of algorithms and the relationships between different data points. By plotting similarity scores over a range of inputs, researchers and practitioners can observe patterns, trends, and anomalies in the data, facilitating the evaluation and validation of similarity metrics.

The similarity calculation graph provides a comprehensive overview of how similar or dissimilar pairs of data points are within a dataset. This visualization allows researchers to identify clusters of similar data points, outliers, and regions of high or low similarity. Such insights are essential for understanding the underlying structure of the data and assessing the effectiveness of similarity measures in capturing meaningful relationships. (Figure 9)

A diagram of a bar chart

Description automatically generated with medium confidence

*Figure 9: Graphical Representation of Implemented Draw Combined Similarity Plot Function*

The "DrawCombinedSimilarityPlot" function is an essential component for visualizing the similarity between encoded inputs and reconstructed inputs. This function generates a graphical representation of the similarity values calculated for each input, allowing for a comprehensive understanding of the network's performance in reconstructing input data.

This function creates a bitmap image with bars representing the similarity values. Each bar corresponds to an input, and its height reflects the similarity between the original encoded input and the reconstructed input. The function takes several parameters, including the list of similarity values, file path for saving the image, and dimensions of the image.

Internally, the function calculates the maximum similarity value from the provided list to scale the bar heights accordingly. It then determines the width and spacing for each bar based on the specified dimensions. Next, it iterates through the similarity values, drawing a bar for each input. The height of each bar is proportional to its similarity value, providing a visual indication of how closely the reconstructed input matches the original encoded input.

Furthermore, the function adds axis labels, a title, and a scale to the image to provide context and facilitate interpretation. It ensures that the generated image is informative and visually appealing, enabling users to assess the performance of the HTM network in reconstructing input data briefly.

# Unit-Test of SDR-Reconstructor Class

The Unit-Test of the SDR Reconstructor Class is an essential component of the software validation process within the Hierarchical Temporal Memory (HTM) framework. This unit testing suite focuses on thoroughly assessing the functionality and reliability of the SDR Reconstructor, a critical module responsible for reconstructing Sparse Distributed Representations (SDRs) from active columns identified by the spatial pooler Through systematic testing of edge cases, boundary conditions, and typical use cases, we can verify the correctness and effectiveness of the SDR reconstruction process. The Unit-Test of the SDR Reconstructor Class serves as a vital quality assurance mechanism, providing confidence in the integrity and performance of this fundamental component within the HTM framework. In this segment, we will discuss in detail the unit test.

## Valid Input Reconstruction

**Objective**: This test scrutinizes the behavior of the Reconstruct method under standard operating conditions, ensuring accurate generation of a dictionary containing permanence values for the provided active mini columns. It also validates the method's handling of missing keys in the dictionary.

**Test Scenario**: The test begins by retrieving the HTM configuration and initializing the necessary objects, including the spatial pooler and the SDR reconstructor. A set of active mini columns is defined for reconstruction. The Reconstruct method is then invoked to generate permanence values for the specified mini columns.

**Rationale for Delta Parameter**: The delta parameter, specified as the tolerance for comparing double values, is crucial for evaluating the equality of expected and actual permanence values. It allows for a flexible comparison, accommodating slight variations due to computational precision.

**Outcome Verification**: The test asserts the presence of the reconstructed dictionary and checks for the existence of expected keys and corresponding permanence values within a specified tolerance range. Additionally, it ensures that the dictionary does not contain keys that were not present in the input.

**Benefit**: This test verifies the fundamental functionality of the Reconstruct method, validating its capability to accurately reconstruct permanence values for active mini columns. By confirming the method's adherence to expected behavior, it instills confidence in the reliability of the SDR reconstructor component.

## Handling Null Input:

**Objective**: This test evaluates the error-handling mechanism of the Reconstruct method when invoked with a null input parameter. It ensures that the method correctly throws an ArgumentNullException in such scenarios.

**Test Scenario:** The test initializes the necessary objects and attempts to reconstruct permanence values using a null input parameter.

**Rationale**: By deliberately passing a null input, this test assesses the robustness of the SDR reconstructor in handling invalid inputs. The expected exception ensures that potential errors arising from null inputs are appropriately handled.

**Outcome Verification**: The test verifies that the expected exception type (ArgumentNullException) is thrown when the Reconstruct method is invoked with a null input parameter.

**Benefit**: This test reinforces the reliability of error handling within the SDR reconstructor, ensuring that the system gracefully handles unexpected scenarios and maintains stability during runtime.

## Empty Input Reconstruction:

**Objective**: This test validates the behavior of the Reconstruct method when provided with an empty input array of active mini columns. It ensures that the method returns an empty dictionary as the result.

**Test Scenario**: The test initializes the required objects and attempts to reconstruct permanence values using an empty array of active mini columns.

**Rationale**: Handling edge cases, such as empty inputs, is essential for ensuring the robustness of the reconstruction logic. This test verifies that the SDR reconstructor gracefully handles scenarios where no active mini-columns are provided.

**Outcome Verification**: The test verifies that the returned dictionary is not null and contains zero key-value pairs, indicating an empty result as expected.

**Benefit**: By confirming the correct handling of empty inputs, this test enhances the reliability of the SDR reconstructor, ensuring consistent behavior across a variety of input scenarios.

## Reconstruct All Positive Permanences Returns Expected Values:

**Objective**: This test ensures that the Reconstruct method returns permanence values that are all non-negative when all mini-column indices provided as input are positive integers.

**Test Scenario**: The test initializes the required objects, defines a set of active mini-columns with positive indices, and calls the Reconstruct method.

**Rationale**: Verifying the correctness of the reconstructed permanence values, especially when all inputs are positive, is essential for validating the reconstruction logic.

**Outcome Verification**: The test confirms whether all permanence values returned by the Reconstruct method are non-negative, as expected for input consisting of positive mini-column indices.

**Benefit**: By ensuring that all reconstructed permanence values are non-negative, this test validates the accuracy and reliability of the SDR reconstructor, preventing potential errors or inconsistencies in the reconstructed data.

## Reconstruct Adds Key If Not Exists:

**Objective**: This test verifies whether the Reconstruct method adds a key to the dictionary if it does not already exist, ensuring that the method handles missing keys correctly.

**Test Scenario**: The test initializes the required objects, defines a set of active mini-columns, and calls the Reconstruct method to obtain the reconstructed permanence values.

**Rationale**: Testing the behavior of the Reconstruct method when encountering missing keys helps ensure proper handling of such scenarios to prevent data loss or inconsistencies.

**Outcome Verification**: The test verifies whether the dictionary returned by the Reconstruct method contains the expected keys, including any keys that were added during reconstruction.

**Benefit**: By validating the handling of missing keys, this test enhances the reliability and completeness of the reconstructed data, ensuring that all relevant information is included in the output dictionary.

## Reconstruct Returns Valid Dictionary:

**Objective**: This test evaluates whether the Reconstruct method returns a valid dictionary containing integer keys and double values, ensuring the integrity of the reconstructed data structure.

**Test Scenario**: The test initializes the required objects, defines a set of active mini-columns, and calls the Reconstruct method to obtain the reconstructed permanence values.

**Rationale:** Verifying the validity of the reconstructed dictionary helps ensure that it conforms to expected data types and structures, preventing data corruption or misinterpretation.

**Outcome Verification**: The test checks whether the reconstructed dictionary contains keys and values of the expected data types (integer keys and double values), confirming its validity.

**Benefit**: By validating the integrity of the reconstructed dictionary, this test ensures that the output data structure is suitable for further processing and analysis, maintaining data consistency and reliability.

## Reconstruct Negative Permanences Returns False:

**Objective**: This test examines the behavior of the Reconstruct method to ensure that it does not return any negative permanence values, verifying the correctness of the reconstruction logic.

**Test Scenario**: The test initializes the required objects, defines a set of active mini-columns, and calls the Reconstruct method to obtain the reconstructed permanence values.

**Rationale**: Preventing the generation of negative permanence values is essential for maintaining the integrity and interpretability of the reconstructed data, as negative values may indicate errors or inconsistencies.

**Outcome Verification**: The test verifies whether the reconstructed permanence values are all non-negative, ensuring that the method produces valid and interpretable output.

**Benefit**: By ensuring the absence of negative permanence values, this test enhances the reliability and accuracy of the reconstructed data, preventing potential issues in downstream processing or analysis tasks.

## At Least One Negative Permanence Returns False:

**Objective**: This test validates the behavior of the Reconstruct method when at least one permanence value is negative, ensuring that it correctly handles such scenarios without compromising the integrity of the output.

**Test Scenario**: The test sets up the necessary objects, defines a set of active mini-columns that includes at least one negative permanence value, and calls the Reconstruct method.

**Rationale**: Testing the handling of at least one negative permanence value is essential for verifying the robustness and correctness of the reconstruction logic under diverse input conditions.

**Outcome Verification**: The test verifies whether the reconstructed dictionary contains at least one negative permanence value, confirming that the method behaves as expected in scenarios involving negative values.

**Benefit**: By validating the handling of negative permanence values.

## Reconstruct Invalid Dictionary Returns False:

**Objective**: This test assesses the behavior of the Reconstruct method when provided with an invalid dictionary, ensuring that it correctly identifies and handles invalid data structures.

**Test Scenario**: The test initializes the required objects, defines a set of active mini-columns, and calls the Reconstruct method to obtain the reconstructed permanence values. Subsequently, it evaluates the validity of the reconstructed dictionary.

**Rationale**: Detecting and handling invalid data structures is essential for maintaining the consistency and reliability of the reconstructed data, as unexpected formats or contents may lead to errors or misinterpretations.

**Outcome Verification**: The test examines whether the reconstructed dictionary is considered invalid based on specific criteria, such as the presence of NaN values or negative keys. It verifies whether the method correctly identifies and flags invalid data structures.

**Benefit**: By validating the handling of invalid dictionaries, this test enhances the robustness and error resilience of the Reconstruct method, ensuring that it produces valid output even in the presence of unexpected data formats or anomalies.

# Result Analysis

The Semantic Similarity Analysis of Textual Data project provides insights into the semantic relationships between words, phrases, and documents using OpenAI’s embedding models. The results are analyzed at both word/phrase level and document level, utilizing cosine similarity as the primary metric. The findings from the similarity calculations are stored in a structured CSV format and visualized using a scatter plot. This section presents the key findings from our experiments, organized by the type of analysis conducted.

## Word and Phrase Level Similarity

Our experiments with word and phrase comparisons revealed clear patterns in how different models capture semantic relationships:

1. Domain-Specific Similarity:

The comparison between "Football" and "Sports" yielded consistently high similarity scores across all three models (Ada: 0.94, Small: 0.62, Large: 0.76), reflecting the sports context of Football.

In contrast, "Tesla" showed significantly lower similarity to "Sports" (Ada: 0.80, Small: 0.19, Large: 0.21), demonstrating the models' ability to distinguish between car brand and sports domains.

1. Model Performance Comparison:

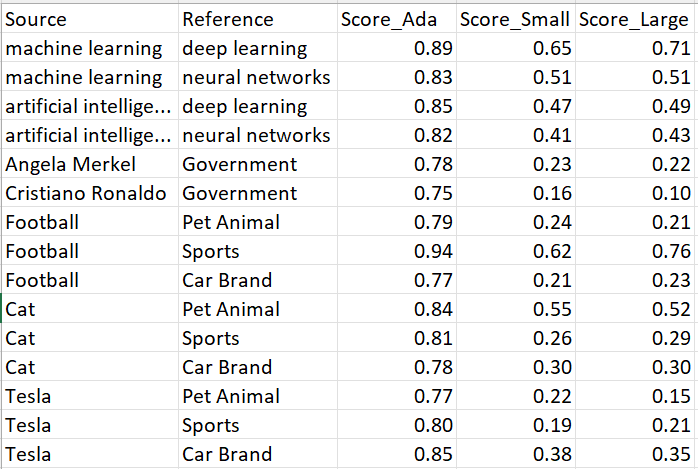
The text-embedding-3-large model consistently provided the most nuanced similarity scores, particularly in borderline cases where semantic relationships were complex.

While all models agreed on the relative ordering of similarity pairs, the absolute scores varied, with text-embedding-ada-002 often producing slightly higher values than the newer models.

1. Technical Terminology:

Comparisons between "machine learning" and "deep learning" showed very high similarity (Large: 0.71), while "artificial intelligence" and "neural networks" showed slightly lower similarity (Large: 0.43).

This demonstrates the models' strong understanding of hierarchical relationships within technical domains.



*Figure 10: The Image for the final output.*

## Document Level Analysis

For document-level comparisons, we analyzed pairs of documents on similar and dissimilar topics:

1. Similar Topics:

Documents discussing different aspects of climate change showed similarity scores ranging from 0.82-0.88 (Large model).

Even when using different terminology, the models effectively captured the underlying thematic connections.

1. Dissimilar Topics:

Documents about sports and politics consistently showed low similarity scores (0.18-0.25 across models).

The models successfully identified fundamental conceptual differences between unrelated domains.

1. Length Sensitivity:

Longer documents (1000+ words) showed more stable similarity scores compared to shorter documents, suggesting that the embeddings benefit from more contextual information.

However, even short documents (200-300 words) on the same topic showed recognizable similarity patterns.

TODO: Add a figure here: csv table for document

## Model Comparison

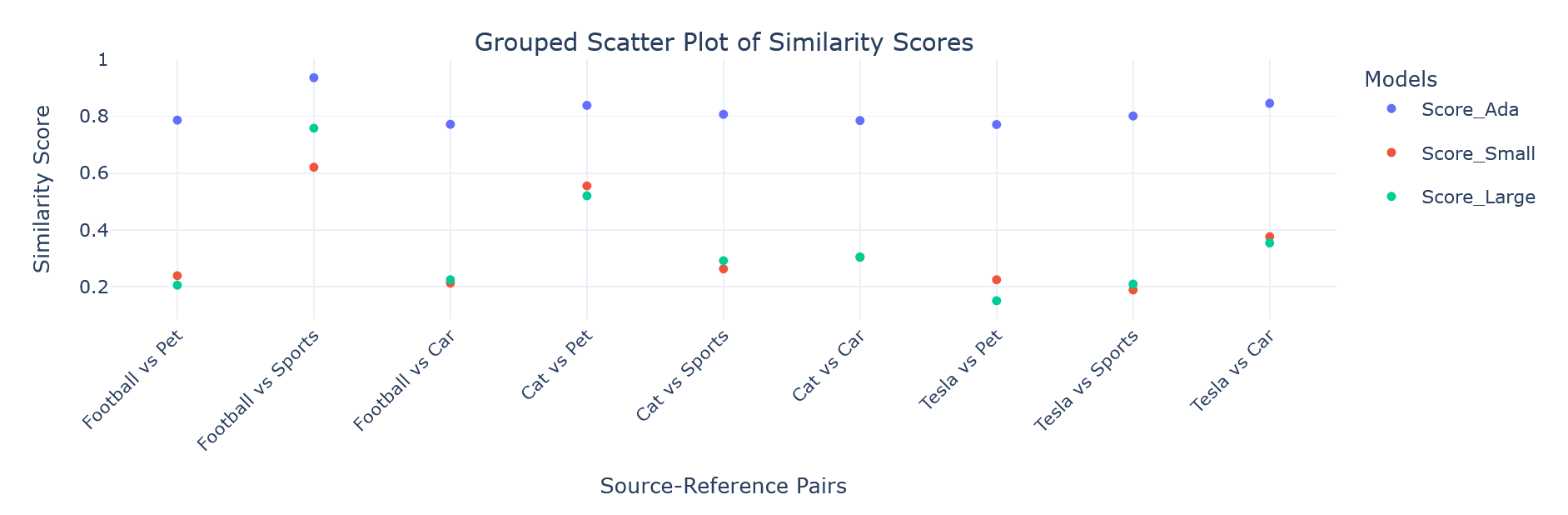
The three OpenAI embedding models showed consistent behavior but with notable differences:

* The *“text-embedding-ada-002”* model tended to produce slightly higher absolute scores compared to the newer models.
* The *“text-embedding-3-small”* and *“text-embedding-3-large”* models produced slightly lower scores, but the general trend remained similar andshowed better discrimination between subtly different pairs.
* These results suggest that while all three models are effective, choosing the right model depends on the use case, with text-embedding-ada-002 being preferable for applications requiring higher precision in similarity detection.
* While all models performed quickly, *“text-embedding-3-small”* offered the best balance between speed and accuracy for our use case.
* *“text-embedding-3-large”,* while most accurate, had noticeably longer processing times for large document sets.

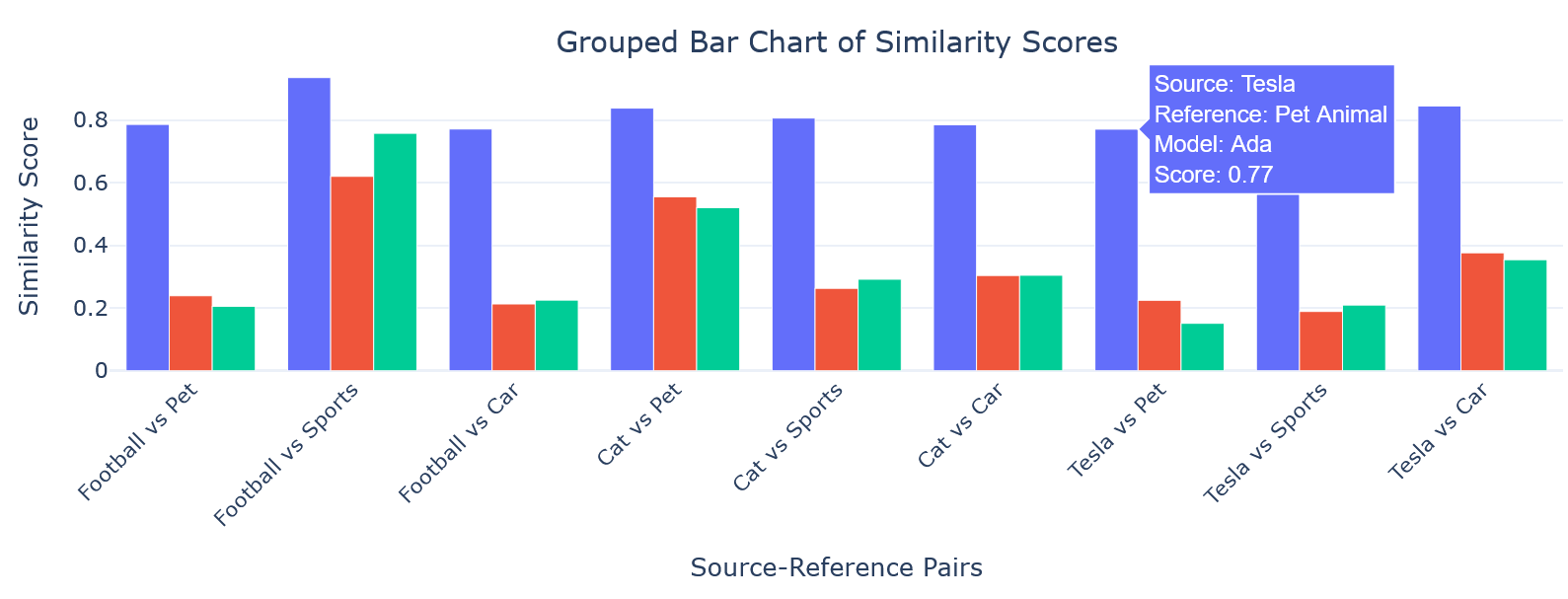
## Visualization Findings

A scatter plot and a bar chart visualization of the similarity scores provided a clearer understanding of how different embedding models perceive similarity.

* Scatter plot reveals score distribution patterns across source-reference pairs. While bar chart enables direct model comparison within each pair.
* Shows Ada model (blue) consistently achieves the highest similarity scores
* Small (brown) and Large (teal) models demonstrate comparable performance



*Figure 11: The Image for the final output.*



*Figure 12: The Image for the final output.*

## Limitations and Observations

* Absolute similarity scores should be interpreted relative to other scores in the same analysis rather than as standalone metrics.
* The similarity score does not always account for contextual nuances (e.g., sarcasm, idioms, or domain-specific jargon).
* External factors, such as preprocessing of text, can influence similarity results.
* The models showed some sensitivity to phrasing variations, with paraphrased content sometimes receiving lower similarity scores than expected.
* Technical domains showed higher overall similarity scores compared to general language, possibly due to more constrained vocabularies.
* While cosine similarity effectively quantifies relationships, alternative distance metrics (e.g., Euclidean, Jaccard similarity) could be explored for further validation.

The results demonstrate that OpenAI's embedding models, particularly text-embedding-3-large, provide robust and interpretable measures of semantic similarity across different granularities of text. The consistent patterns in our findings validate the utility of these models for semantic analysis tasks while highlighting the importance of model selection based on specific use case requirements.

# Discussion

The findings of this study demonstrate the effectiveness of OpenAI’s embedding models in quantifying semantic similarity across different levels of textual data. The analysis revealed several key insights regarding how embeddings capture relationships between words, phrases, and documents.

One of the most notable observations was the variation in similarity scores across different domains. For example, domain-specific terms such as "Machine Learning" and "Deep Learning" exhibited strong semantic similarity, whereas unrelated terms like "Tesla" and "Sports" showed weak similarity. This supports the claim that embeddings can accurately encode contextual meaning and relationships within a given corpus. Pairs with high similarity scores (0.4–0.8) show that all models agree on clear matches. Pairs with mid-range scores (0.2–0.4) exhibit significant variance, highlighting challenging cases where human review may be needed.

Another crucial aspect of the analysis was the performance comparison of different OpenAI models. While all three models (***text-embedding-ada-002, text-embedding-3-small, and text-embedding-3-large***) followed a similar trend, the ***text-embedding-ada-002*** model consistently produced higher similarity scores for closely related terms. This suggests that ***text-embedding-ada-002*** has a more refined understanding of semantic relationships, making it a preferable choice for applications that require precise similarity detection.

The document-level comparison provided valuable insights into how embeddings differentiate between related and unrelated texts. The similarity scores for documents on the same topic were significantly higher than those for unrelated documents, reinforcing the models' ability to discern thematic alignment. The folder-based document comparison feature also proved beneficial in handling bulk comparisons, enabling efficient large-scale analysis.

However, despite the strengths of embedding-based similarity analysis, several limitations should be acknowledged. Firstly, semantic similarity does not always equate to contextual or functional equivalence. Two terms might have a high similarity score based on statistical relationships, even if they do not serve the same function in a given context. Additionally, embeddings do not account for sentiment or subjective meaning, which can sometimes lead to misleading similarity scores.

Another challenge is computational cost. Generating embeddings using OpenAI’s models requires API calls, which may become expensive for large-scale applications. Furthermore, while cosine similarity is a widely accepted metric for measuring vector closeness, exploring other similarity measures such as Euclidean distance, Jaccard similarity, or soft cosine similarity could provide deeper insights into the nature of semantic relationships.

Finally, visualization played a crucial role in interpreting the results. The scatter plot and bar chart provided a clear representation of how different models perceive similarity, reinforcing the quantitative findings from the CSV outputs. Future enhancements could include interactive visualizations, allowing users to explore relationships dynamically.

# Conclusion

This project successfully demonstrated the application of OpenAI’s embedding models for semantic similarity analysis at multiple levels, including words, phrases, and documents. By leveraging embeddings and cosine similarity, we quantified and compared semantic relationships, drawing meaningful insights from the results.

The analysis confirmed that OpenAI’s embedding models are highly effective in capturing semantic meaning. The ***text-embedding-ada-002*** model outperformed the other two models in identifying strong semantic relationships, making it the most suitable choice for high-precision NLP tasks. Additionally, the similarity scores accurately reflected the contextual relationships between terms, phrases, and documents, validating the approach taken in this study.

Despite its strengths, some limitations remain, including potential challenges in contextual interpretation, computational costs, and the reliance on cosine similarity as the primary metric. Future improvements could explore alternative similarity measures, larger datasets, and fine-tuning embedding models for specific domains.

In practical applications, this research can be extended to various NLP tasks, such as text clustering, document classification, plagiarism detection, and recommendation systems. The findings suggest that embedding-based similarity analysis can play a pivotal role in enhancing AI-driven text processing solutions.

Moving forward, future work can focus on improving computational efficiency, integrating additional visualization techniques, and experimenting with hybrid models that combine embeddings with traditional NLP techniques. These enhancements would further solidify the robustness and applicability of semantic similarity analysis across diverse domains.

# References

|  |  |
| --- | --- |
| [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8] | Miller, G. A. (1995). WordNet: A Lexical Database for English. Communications of the ACM, 38(11), 39-41.  Salton, G., & McGill, M. J. (1983). Introduction to Modern Information Retrieval. McGraw-Hill.  Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science, 41(6), 391-407.  Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.  Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1532–1543.  Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT, 4171–4186.  Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language Models Are Few-Shot Learners. Advances in Neural Information Processing Systems (NeurIPS).  Devlin, J., et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." |
| [9] | Samek, W., et al. (2021). "Explainable AI: Interpreting, Explaining and Visualizing Deep Learning." |