Semantic Analysis of Textual Data

line 1: 1st Given Name Surname   
line 2: email address

line 1: 2nd Given Name Surname  
line 2: email address

line 1: 3rd Given Name Surname  
line 2: email address

*Abstract— In the era of big data and natural language processing, the ability to accurately analyze and compare textual data is paramount. This research paper presents a comprehensive framework for semantic analysis of textual data, focusing on the calculation of phrase similarities and document comparisons. Leveraging advanced open AI embedding techniques and Cosine Similarity Algorithms, our approach aims to analyses the accuracy and efficiency of semantic similarity assessments and understands its potential use case applications. The framework is implemented as a software tool that preprocesses textual data, generates embedding’s using state-of-the-art models, and calculates similarity scores between phrases and documents. The tool supports various preprocessing options, hence ensuring robust and contextually relevant similarity measurements. We evaluate the performance of our framework through extensive experiments on diverse datasets, demonstrating its effectiveness in capturing semantic nuances and improving the quality of textual data analysis. To visualize the effectiveness of our similarity calculations, we have developed a plotting tool that maps scalar values against similarity scores allowing for a clear and immediate understanding of the data distribution and the performance of our similarity metrics and where secondary plot displays the number of possible comparisons between documents and its corresponding Similarity score which displays the semantic analysis between documents or phrases with its relevance based on pre-defined threshold. The results indicate significant improvements in similarity scoring accuracy compared to traditional methods. This research contributes to the field of natural language processing by providing a scalable and flexible solution for semantic analysis, with potential applications in automated content categorization like resume filtering for relevant job opportunities or filtering admission for students based on admission requirements and other several use cases across different domains. The software tool is made available as an open-source project, encouraging further research and development in this analysis or develops potential use cases.*

*Keywords—Embedding, Cosine Similarity, scalar*

# Introduction

Billions of brief text messages are sent on social media every day; according to statistics, nearly every tweet is between one and thirty words long. Appropriate information retrieval methods are required in order to access this stream of really brief text fragments. Tf-idf (Term Frequency-Inverse Document Frequency) is a well-known and conventional representation for comparing texts, including news articles, with one another. This method calculates a term's frequency within the document and its inverse document frequency (IDF) throughout the entire corpus, hence identifies a term's significance in a document, but it frequently fails as it depends on word overlap to discover similarities. This demands the requirement for sentence representations that understand more than just word contents rather a semantic similarity [1].

Semantic similarity is a key component of Natural Language Processing (NLP) and one of the core tasks for many NLP applications and related fields. Semantic similarity, as opposed to the lexicographical similarity or the above mentioned statistical similarity, is a metric that is defined over a collection of documents or phrases. It bases the concept of distance between objects on how similar their meanings or semantic content are. Also, similarity between the documents is based on the direct and indirect relationships among them, which can be measured and identified by the presence of semantic relations among them. In the realm of Natural Language Processing (NLP), estimating the semantic similarity between text data is one of the most difficult and unresolved research challenges. It is challenging to create rule-based techniques for calculating semantic similarity metrics due to the flexibility of natural language. Numerous semantic similarity techniques have been put out over time to address this problem [2]. To be precise, semantic similarity, which quantifies how closely two pieces of text align in meaning, is a fundamental concept in natural language processing (NLP) with applications in information retrieval, document clustering, and recommendation systems. By utilising OpenAI's powerful embeddings, we can turn text into dense vector representations that capture its semantic essence, allowing us to compute similarity metrics like cosine similarity.

In this paper, we have designed a systematic approach to investigate and measure the semantic relationships between textual data at different levels, ranging from individual words and phrases to entire documents. To make this possible we have used OpenAI's GPT-based embeddings in order generate embeddings based on respective context. To maintain maximum system accuracy, we have also incorporated a preprocessing module and the raw inputs are first fed to the preprocessor, which minimizes the words, still maintaining the context. This is done in order to optimize the token usage, since GPT is not free. Once, the embedding is generated, the system performs similarity analysis to determine how different textual elements relate to one another in terms of meaning, context, and domain. The calculated score is later written into a CSV file to generate the visualization.

Words/Phrases

n

Preprocessor

Documents

Embedding Generation

Cosine Method

Similarity Calculation

CSV File Generation

Python

Visualization

*Fig 1: Simplified Block Diagram*

As shown in Fig 1; an external python module is used to generate the visualization part successfully, we used this external source inorder to**…………………..???**

# Methods

## Literature Review

Semantic analysis of textual data aims to extract valuable insights from text by understanding the underlying structure and meaning of words, phrases, and texts. This process has become more important for applications such as retrieving information, document classification, sentiment analysis, and natural language understanding. Many approaches and frameworks have evolved throughout time to improve semantic analysis. The emergence of transformer-based models such as GPT has further speed up progress in this field. However, this sector continues to benefit greatly from a variety of modern and old methodologies. The below section explores both traditional and distributional, and also the switching into the GPT based contemporary approach.

# A.1 Classical Methods for Semantic Analysis

One fundamental method that uses statistical calculations to identify connections between words and documents is called Latent Semantic Analysis (LSA). LSA is a natural language processing technique that generates a set of concepts associated with a set of documents and terms by analysing the links between the documents and terms. It facilitates the discovery of the data's underlying latent semantic structure.

Singular Value Decomposition (SVD) is used to break down a term-document matrix, lowering the dimensionality and emphasising the latent structure in the data **[3].** LSA has been effectively used in information retrieval, topic modelling, and document similarity. Its dependence on a linear translation, however, restricts its capacity to represent intricate contextual connections among words.

In contrast to LSA, Explicit Semantic Analysis (ESA) creates high-dimensional representations of text by using structured knowledge sources such as Wikipedia [4]. ESA can effectively determine semantic relatedness by mapping text to a concept space. While ESA benefits from leveraging external expertise, its effectiveness mostly depends on the depth of the underlying knowledge base.

# A.2 Distributional Semantic Models

With word embeddings, a significant breakthrough in capturing the semantic links between words was achieved. In order to produce dense vector representations, Word2Vec [5] and GloVe [6] evaluate the context of words in large datasets. Although these models are successful in capturing semantic similarity, they are limited by their inability to accurately represent context-dependent interpretations and polysemy.

On the other hand, sentence embeddings are designed to give full phrases or documents a meaningful vector space representation. To generate sentence embeddings that score well on similarity and clustering tests, a transformer-based architecture known as Sentence-BERT (SBERT) [7] was introduced. DeCLUTR (Deep Contrastive Learning for Unsupervised Textual Representations) creates unsupervised sentence representations through contrastive learning that exhibit remarkable performance in several downstream tasks. In comparison to traditional word embeddings, these models provide context-aware representations.

Since the introduction of transformer systems, semantic analysis has changed. By considering the context in which words appear, models such as BERT (Bidirectional Encoder Representations from Transformers) provide contextual embeddings that improve on previous methods [8]. Transformer-based models have demonstrated exceptional performance in a variety of natural language processing tasks, including text summarisation, sentiment analysis, and question answering. An efficient method for producing superior sentence embeddings is contrastive learning. DeCLUTR performs well on problems involving similarity by using contrastive learning to unsupervised textual representations. When labelled data is hard to come by or unavailable, this method is especially helpful.

## Evolution to OpenAI’s GPT Model for Semantic Analysis

In our project we have utilized the Open AI’s GPT model to enhance semantic analysis procedure in a flexible and robust way. This transition from distributional and classical approaches to transformer-based designs such as GPT is due to the following reasons and this, in turn represents a paradigm change in semantic analysis.

i) LSA and ESA cannot capture contextual meaning because they rely on statistical connections instead of true semantic understanding.   
ii) Word Embeddings (Word2Vec, GloVe) generate static embeddings that guarantee that words are vectorially represented consistently across contexts. Accurate analysis of polysemy is impossible due to this limitation.  
iii) Because Sentence Embeddings (SBERT, DeCLUTR) are context-aware, they require extensive training and fine-tuning optimisations, which can be resource-intensive.

The meaning of words, sentences, and texts is accurately captured via dynamic, context-sensitive embeddings created using an OpenAI model based on GPT. With just a basic preprocessor module, these embeddings can be used straight away without a lot of fine-tuning or training.   
Additionally, because GPT embeddings are built using the complete context of the text, they effectively capture polysemous meanings. The substantial pre-training on text enables the model to increase its generalisation across other domains, providing strength and versatility.

Numerous methods have been used to approach semantic analysis, each with its own advantages and disadvantages. Conventional methods such as LSA and ESA are simple and intuitive, but they lack the contextual sensitivity of embedding-based methods. Word and phrase embeddings provide more meaningful representations, but they struggle with polysemy. Transformer-based models solve many of these problems, despite the fact that they may require a lot of resources.

# *B.1 Embedding model:*

An embedding is a list of floating-point numbers that is vectorised. To be precise, it is a series of numbers that represent the ideas in content, like code or natural language. Machine learning models and other algorithms can hence easily comprehend the connections between content and carry out tasks such as retrieval or clustering based on these embeddings. They represent a variety of input formats that machine learning models can interpret, including text, photos, and audio. Tokenising text into tokens is the first step an AI model takes after receiving text input. After that, each token is transformed into its matching embedding. They power various retrieval augmented generation (RAG) developer tools and applications such as knowledge retrieval in ChatGPT and the Assistants API. Two vectors' distance from one another indicates how related they are. Large distances indicate low relatedness, while small distances indicate high relatedness. [9][10].

The below figure, shows how different words/phrases are mapped into a high dimensional vector space.

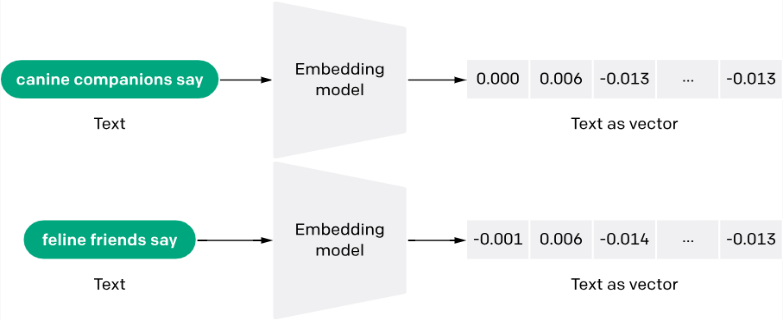


FIG:

Here, an embedding model turns each text into vectors. For instance, the short phrase "anatine amigos" is transformed into a single, large vector (e.g., 1536 dimensions for text-embedding-ada-002), with each dimension capturing a distinctive aspect or feature of the text's meaning. The goal is to represent the text's semantic meaning in a multi-dimensional space, where similar phrases have closer vectors. This process is crucial for semantic similarity analysis, where the closeness of the vectors indicates semantic similarity. It should be noted that, instead of being chosen at random, these vectors are intended to encode the text's meaning such that related sentences will have similar vectors.

Consider the JSON response from the OpenAI API for generating embeddings shown below:

{

"object": "list",

"data": [

{

"object": "embedding",

"index": 0,

"embedding": [

-0.006929283495992422,

-0.005336422007530928,

-4.547132266452536e-05,

-0.024047505110502243

]

}

],

"model": "text-embedding-3-small",

"usage": {

"prompt\_tokens": 5,

"total\_tokens": 5

}

Here, it is given that the prompt\_tokens is 5.   
This shows how many tokens were used in the given input text after the model processed it. Tokens are nothing but pieces of characters or words.  
For instance in the above fig 1; “canine companions say”  
" canine, companions and say " are three tokens that could be separated from the phrase " canine companions say."  
Nevertheless, the model may further split it if it employs a method called sub-word tokenisation, such as:

"canine" → ["ca", "nine"]

"companions"→ ["comp", "anions"]

“say” → [“say”]

Similarly, total\_tokens is also 5. This indicates how many tokens the model has handled overall.  
When it comes to embeddings, total\_tokens and prompt\_tokens are typically equal because here we are not producing more text, instead   
only a vector representation of the input text is being encoded. These two concepts are very important in terms of   
cost calculation and management, as OpenAI generate bills according to how many tokens it processes.  
 and also for performance tracking. Therefore, if divided into smaller parts, the phrase "Anatine amigos" may have a total token count of 5.

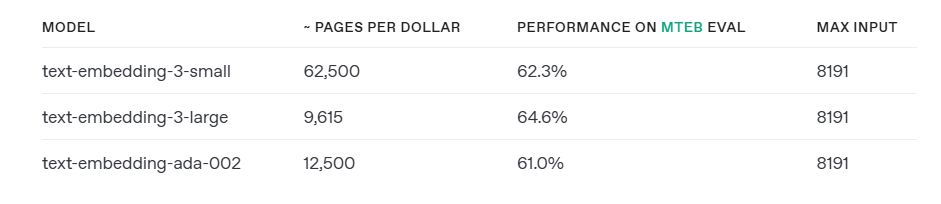
OpenAI provides two robust third-generation embedding models, which are indicated by the model ID's -3  
  
  


FIG:

From the figure, the embedding model is not free. There is always a token limit per users. An incredibly effective embedding model, text-embedding-3-small, is a major improvement over the text-embedding-ada-002 model, which was launched in December 2022⁠. Additionally, text-embedding-3-small is far more effective than our text-embedding-ada-002 model from the previous generation. Also, text-embedding-3-small's price has been lowered from $0.0001 per 1,000 tokens to $0.00002, a 5X reduction from text-embedding-ada-002. Apart from this the new, next-generation larger embedding model, text-embedding-3-large, can produce embeddings up to 3072 dimensions in size. The price of text-embedding-3-large is set at $0.00013 per 1,000 tokens. However, while choosing embedding model, the size of the model should be selected appropriately. Larger embeddings (such as text-embedding-3-large, which has dimensions of 3072).  
This gives text representations that are more precise and thorough and also improved performance on challenging NLP tasks such as text retrieval and document similarity.  
But this requires more compute, memory, and storage expenses. While embeddings that are smaller (such as text-embedding-3-small or reduced versions of larger embeddings) is quicker and works at a lower cost for generating embeddings. Additionally, reduce the amount of memory and storage. This can be avoided by passing a “dimensions” parameter to the model, which can remove some numbers at the end of vectors still maintain the accuracy [9]. This dimensions API parameter allows developers to specify the desired embedding size, hence optimized usage.

When working with natural language and code, embeddings are helpful since they are easily absorbed and compared by

various machine learning models and algorithms, such as search or clustering. Semantically related embeddings are likewise numerically similar. In the below figure, when it comes to "canine companions say," for instance, the embedding vector will resemble "woof" more than "meow." Assume, a dimension is represented by each box with floating-point integers, and each dimension is associated with a characteristic or quality that may or may not be understandable by humans. While more complex data models may contain tens of thousands of dimensions, large language model text embeddings usually have a few thousand.   
Due to the similarities and variations in the meaning of the two words, certain of the dimensions of the two vectors in the example above are comparable, while other dimensions are different [11]

.  
This figure below illustrates the spatial proximity of comparable vectors and contrasts them with significantly dissimilar vectors. The depth and variation of similarity can be estimated by using a suitable distant function:

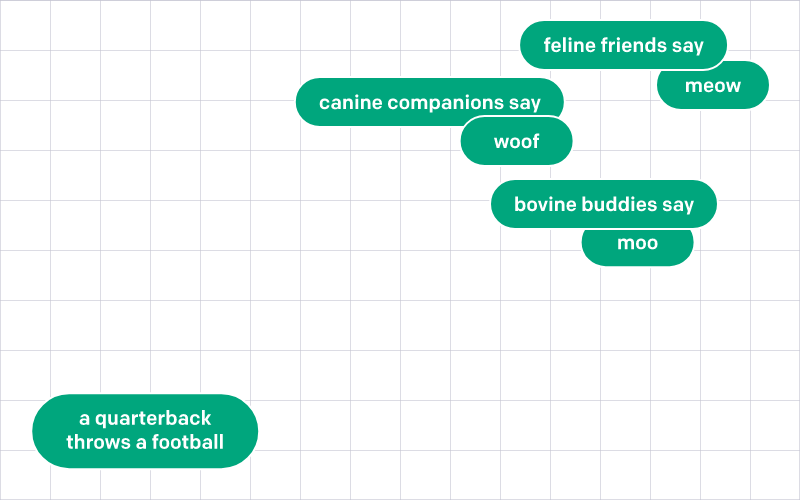


FIG:

## Similarity Analysis- A theoretical estimation about Distant Metrics

The degree to which two objects are similar is measured by their similarity. In the context of data mining, a similarity measure is a distance whose dimensions correspond to the attributes of the objects. Two items are quite similar if they are close together, as well as a low degree of similarity if they are far apart. Mathematical formulas called distance functions are used to quantify how similar or dissimilar two vectors are. The Manhattan distance, Euclidean distance, cosine similarity, and dot product are typical examples. In order to ascertain the degree of relationship between two pieces of data, these metrics are essential [12].

A distance function or metric is a function d(x,y)   
that uses a non-negative real number to quantify the distance between a set's elements. Under that particular measure, the items are equal if the distance is 0. Thus, distance functions give us a mechanism to quantify the proximity of two

elements, which can be vectors, matrices, or any other kind of object. Distance functions are frequently employed in optimisation problems as cost or error functions that need to be minimised.

# Results

This Part of the text describes results of your works. There can only be mentioned references, MUST point back to Methods and Intro chapter. No more external references.

Code examples must be provided to demonstrate how to use the algorithm/module. Provide a reference to more unit tests, which show the same in more detail. Also provide all diagrams with comments and reference to unit tests, which generate diagrams.

# Discussion

Conclusion of your work should be precise and concise. How was the project, what is done, what is the result... There can be discussion on further work and direction.

# Ease of Use

## Selecting a Template (Heading 2)

First, confirm that you have the correct template for your paper size. This template has been tailored for output on the A4 paper size. If you are using US letter-sized paper, please close this file and download the Microsoft Word, Letter file.

## Maintaining the Integrity of the Specifications

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

# Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections A-D below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

## Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

## Units

* Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as “3.5-inch disk drive”.
* Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
* Do not mix complete spellings and abbreviations of units: “Wb/m2” or “webers per square meter”, not “webers/m2”. Spell out units when they appear in text: “. . . a few henries”, not “. . . a few H”.

may use the solidus ( / ), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

*a**b* 

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use “(1)”, not “Eq. (1)” or “equation (1)”, except at the beginning of a sentence: “Equation (1) is . . .”

## Some Common Mistakes

* The word “data” is plural, not singular.
* The subscript for the permeability of vacuum **0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter “o”.
* In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
* A graph within a graph is an “inset”, not an “insert”. The word alternatively is preferred to the word “alternately” (unless you really mean something that alternates).
* Do not use the word “essentially” to mean “approximately” or “effectively”.
* In your paper title, if the words “that uses” can accurately replace the word “using”, capitalize the “u”; if not, keep using lower-cased.
* Be aware of the different meanings of the homophones “affect” and “effect”, “complement” and “compliment”, “discreet” and “discrete”, “principal” and “principle”.
* Do not confuse “imply” and “infer”.
* The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
* There is no period after the “et” in the Latin abbreviation “et al.”.
* The abbreviation “i.e.” means “that is”, and the abbreviation “e.g.” means “for example”.

An excellent style manual for science writers is [7].

# Using the Template

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

## Authors and Affiliations

**The template is designed for, but not limited to, three authors.** A minimum of one author is required for all report articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

### For papers with more than three authors: Add author names horizontally, moving to a third row if needed for more than 8 authors.

### For papers with less than three authors: To change the default, adjust the template as follows.

#### Selection: Highlight all author and affiliation lines.

#### Change number of columns: Select the Columns icon from the MS Word Standard toolbar and then select the correct number of columns from the selection palette.

#### Deletion: Delete the author and affiliation lines for the extra authors.

## Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1”, “Heading 2”, “Heading 3”, and “Heading 4” are prescribed.

## Figures and Tables

For adding object other than text (tables, equations, graphs, figures, code…), **there must be at least one cross reference** to it. Figure 1 is an example

#### Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

1. Table Type Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. (*Table footnote*)



Figure 1 Example Figure Caption

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

## Code References:

Referencing Code in your text should be avoided unless necessary. In such cases it can be inserted as a listing as shown in **Error! Reference source not found.**

Listing 1 Code Reference Example

Console.WriteLine(“Referencing code”, var);

// using tab can be replaced with 4 spaces

Do not pass code as image. When referring to variable in **Error! Reference source not found.**, italics should be used for example *var.* Code flows and logic should be presented better as Graph or Diagram instead of words.

Code Block which is too big to put in the textbox can be reference as Listing 2.

Listing 2 Unit Test [EncodeDateTimeTest](https://github.com/ddobric/neocortexapi/blob/0348ffb99739ddf8c8c3a875f8162a18073938ca/source/UnitTestsProject/EncoderTests/DateTimeEncoderExperimentalTests.cs#L34-L49)

public void EncodeDateTimeTest(int w, double r, …)

{

…

DateTimeEncoderExperimental encoder = new…

var result = encoder.Encode(input);

…

Assert.IsTrue(result.SequenceEqual(expected…

}

##### Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

##### References

[1] S. V. C. S. B. T. D. a. B. D. C. De Boom, “Learning Semantic Similarity for Very Short Texts,” in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Atlantic City, NJ, USA, 2015.

[2] P. P. A. G. D. P. Goutam Majumder, “Semantic Textual Similarity Methods, Tools, and Applications: A Survey,” Computacion y Sistemas , 2016.

[3] S. T. D. G. W. F. T. K. L. R. H. Scott Deerwester, “Indexing by latent semantic analysis,” *Journal of the American Society for Information Science,* vol. 41, no. 6, pp. 391-407, 1990.

[4] S. M. Evgeniy Gabrilovich, “Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis,” *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence,* vol. 6, pp. 1606-1611, 2007.

[5] K. C. G. C. J. D. Tomas Mikolov, “Efficient Estimation of Word Representations in Vector Space,” in

International Conference on Learning Representations, 2013.

[6] R. S. C. M. Jeffrey Pennington, “GloVe: Global Vectors for Word Representation,” in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014.

[7] I. G. Nils Reimers, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)", Hong Kong, China, Association for Computational Linguistics, 2019, pp. 3982--3992.

[8] M.-W. C. K. L. K. T. Jacob Devlin, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota, Association for Computational Linguistics, 2019, pp. 4171--4186.

[9] <https://openai.com/index/new-embedding-models-and-api-updates/>

[10] <https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/vector-embeddings>

[11] https://openai.com/index/introducing-text-and-code-embeddings/

[12] N. K. K. ,. K. S. Dipendra Prasad Yadav, “Distance Metrics for Machine Learning and it's Relation with Other Distances.,” Mikailalsys Journal of Mathematics and Statistics, vol. 1, pp. 15-23, 2023.