

**WORD EMBEDDING FOR TAMIL LANGUAGE WITH VISUALIZATION**

*Submitted by*

**NITHISHVAR K (221501089)**

**PADMACHARAN K (221501091)**

**AI19643 FOUNDATIONS OF NATURAL LANGUAGE PROCESSING**

Department of Artificial Intelligence and Machine Learning

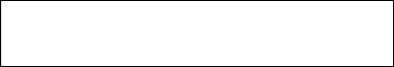
Rajalakshmi Engineering College, Thandalam

****

**BONAFIDE CERTIFICATE**

**NAME ……………………………………………………………………………….…….…**

**ACADEMIC YEAR…………….………SEMESTER………….BRANCH………………**



**UNIVERSITY REGISTER No.**

Certified that this is the bona fide record of work done by the above students in the Mini Project titled **"WORD EMBEDDING FOR TAMIL LANGUAGE WITH VISUALIZATION”** in the subject **AI19643 FOUNDATIONS OF NATURAL LANGUAGE PROCESSING** during the year **2024 - 2025.**

**Signature of Faculty – in – Charge**

Submitted for the Practical Examination held on

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ABSTRACT**

Embedding words into vectors is one of the key techniques in Natural Language Processing (NLP) as it allows transformation of words into vectors capturing information about its meaning and relationships with other words. This paper addresses the process of generating word embeddings for the Tamil language – a/Classical South Indian language that is morphologically rich and is characterized by complex structure. Given the scarce resources available for the language, constructing quality embeddings for Tamil poses an interesting problem. This research applies unsupervised methods like Word2Vec, GloVe, and FastText to a corpus of Tamil language literary works, news articles, and content available online. Traditional models sub-learn typography of various languages fail to capture the complexities of Tamil morphology; thus, FastText shines due to its subword level modeling.

The embeddings are evaluated using intrinsic methods consisting of performing word similarity tasks and visualizing the results with dimensionality reduction techniques such as t-SNE or PCA. Visualization permits examination of names such as gender pairs and list of synonyms which can be grouped together under the same topic. All these demonstrate that the embeddings provide sufficient details about the language.

TAMIL proved to be effective for the characterization of linguistic phenomena within advanced Tamil. Therefore, comparing different best models within a template showed that Fast Text gave better precision results.

***Keywords:***

*Tamil NLP, Word Embedding, Word2Vec, FastText, GloVe, Morphologically Rich Language, Semantic Representation, Tamil Corpus, Vector Space Visualization.*

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** |  |
| 1. | **INTRODUCTION** | 1 |
| 2. | **LITERATURE REVIEW** | 3 |
| 3. | **SYSTEM REQUIREMENTS**  3.1HARDWARE REQUIREMENTS  3.2 SOFTWARE REQUIREMENTS | 8 |
| 4. | **SYSTEM OVERVIEW** | 9 |
| 4.1 EXISTING SYSTEM  4.1.1 DRAWBACKS OF EXISTING SYSTEM |  |
| 4.2 PROPOSED SYSTEM  4.2.1 ADVANTAGES OF PROPOSED SYSTEM | 10 |
| 5 | **SYSTEM IMPLEMENTATION**  5.1 SYSTEM ARCHITECTURE DIAGRAM | 11 |
| 5.2 SYSTEM FLOW | 12 |
| 5.3 LIST OF MODULES | 13 |
| 5.4 MODULE DESCRIPTION | 14 |
| 6 | **RESULT AND DISCUSSION** | 15 |
| 7 | **APPENDIX** | 20 |
| SAMPLE CODE |
| OUTPUT SCREENSHOTS |
|  | **REFERENCES** | 30 |

**CHAPTER 1**

**INTRODUCTION**

Word embeddings-methods that map words into continuous vector spaces where semantic and syntactic relationships are preserved—have been a major development in Natural Language Processing (NLP) in recent years. Numerous NLP applications, including text classification, sentiment analysis, machine translation, and information retrieval, rely on these embeddings as fundamental components. Many Indian languages, including Tamil, are still underrepresented in this field, whereas languages like English benefit from an abundance of annotated corpora and pre-trained models. Because of its intricate word formations, inflections, and syntactic variations, Tamil, a classical language with rich morphology and agglutinative grammar, presents special difficulties for conventional NLP models.

Natural Language Processing (NLP) has witnessed significant advancements in recent years, particularly through the development of word embeddings—techniques that map words into continuous vector spaces where semantic and syntactic relationships are preserved. These embeddings serve as foundational elements in a wide range of NLP applications such as text classification, sentiment analysis, machine translation, and information retrieval. While languages like English benefit from a wealth of annotated corpora and pre-trained models, many Indian languages, including Tamil, remain underrepresented in this space. Tamil, a classical language with rich morphology and agglutinative grammar, poses unique challenges for traditional NLP.

This study focuses on building effective word embeddings for the Tamil language using widely adopted models such as Word2Vec, GloVe, and FastText. Among these, FastText is particularly well-suited for morphologically rich languages like Tamil as it considers subword information, enabling better handling of compound and rare words.

To evaluate and interpret the quality of these embeddings, visualization techniques such as t-distributed Stochastic Neighbor Embedding (t-SNE) and Principal Component Analysis (PCA) are employed. These methods reduce high-dimensional vectors into two or three dimensions, making it possible to observe semantic groupings and relationships among Tamil words. Such visualizations not only validate the embedding models but also provide valuable insights into the structure and meaning of Tamil language constructs.

Word embeddings have become a cornerstone of modern Natural Language Processing (NLP), providing a way to represent words as dense vectors that capture their semantic and syntactic relationships. While these techniques have shown great success in widely spoken languages like English, low-resource languages such as Tamil have not seen equivalent advancements due to limited linguistic datasets and tools. Tamil, one of the oldest and most morphologically rich languages in the world, presents unique challenges for NLP due to its agglutinative grammar, extensive inflections, and complex word forms. This study focuses on developing effective word embeddings for the Tamil language using models such as Word2Vec, GloVe, and FastText. FastText, in particular, is well-suited to handle Tamil’s rich morphology through subword-level modeling. To evaluate and better understand the learned embeddings, dimensionality reduction techniques like t-SNE and PCA are applied for visualization, revealing meaningful word groupings and semantic patterns. By addressing the challenges of Tamil language representation and offering visual insights into word relationships, this work contributes to the broader goal of building robust NLP tools for underrepresented languages and supporting various downstream applications such as text classification, sentiment analysis, and machine translation.

Word embedding is a powerful technique in Natural Language Processing (NLP) that represents words as numerical vectors, capturing their meaning and context. While widely used in English and other major languages, Tamil—a classical, morphologically rich South Indian language—has limited NLP resources. This study aims to develop word embeddings for Tamil using models like. FastText is especially effective due to its ability to model subword information, which suits Tamil’s complex word structures. A large Tamil text corpus is used to train the models. To evaluate and interpret the embeddings, visualization techniques such as t-SNE and PCA are applied. These help in understanding word relationships and grouping similar terms. The results contribute to improving Tamil language tools for tasks like translation, sentiment analysis, and text classification.

**CHAPTER 2**

**LITERATURE REVIEW**

[1] Title: BERT: Pre-training of Deep Bidirectional Transformers for Language

Author: Devlin, J., Chang, M. W., Lee, K., & Toutanova, K.

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model that revolutionized natural language processing (NLP). Its key innovation was pre-training deep bidirectional representations from unlabeled text by jointly conditioning on both the left and right context in all layers. This bidirectionality is achieved through two unsupervised prediction tasks: Masked Language Modeling (MLM), where some tokens in the input are masked and the model tries to predict them, and Next Sentence Prediction (NSP), where the model predicts whether a second sentence is the subsequent such as text classification, question answering, and natural language inference, achieving state-of-the-art results on many benchmarks.

[2] Title: RoBERTa: A Robustly Optimized BERT Pretraining Approach

Author: Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019).

RoBERTa (Robustly Optimized BERT Pretraining Approach) is an improved version of BERT that focuses on enhancing the pre-training procedure. The authors found that the Next Sentence Prediction (NSP) objective was not always beneficial and decided to remove it. They also trained the model for longer, with larger batch sizes, and on a significantly larger and more diverse dataset, including BookCorpus, CC-News, OpenWebText, and Stories. These modifications led to substantial performance gains on various NLP tasks compared to the original BERT model. While RoBERTa achieved better results, it retained the computationally intensive nature of BERT and, due to the removal of NSP, might face challenges in tasks explicitly requiring sentence-level relationship understanding.

[3] Title: XLNet: Generalized Autoregressive Pretraining for Language Understanding

Author: Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov.

XLNet aimed to address some limitations of BERT's pre-training method. BERT's masking approach ignores the dependencies between the masked tokens. XLNet, on the other hand, utilizes a permutation language modeling objective. It randomly permutes the order of tokens in the input sequence and then uses an autoregressive model to predict the original sequence. This allows XLNet to capture bidirectional contexts without relying on masking. It was pre-trained on very large datasets, including Wikipedia, BookCorpus, Giga5, ClueWeb09, and Common Crawl, and achieved state-of-the-art results on several benchmarks. However, the permutation-based training process introduces complexity in implementation and significant computational overhead.

[4] Title: ALBERT: A Lite BERT for Self-supervised Learning of Language Representations

Author: Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R.

ALBERT (A Lite BERT) was designed to create a more parameter-efficient version of BERT without significantly sacrificing performance. It introduces two main techniques to reduce the model size and improve training speed: factorized embedding parameterization, which decomposes the large vocabulary embedding matrix into two smaller matrices, and cross-layer parameter sharing, where the parameters of all encoder layers are shared. ALBERT was trained on the same data as BERT but with additional data augmentation. It achieved comparable or even better performance than BERT on several tasks with a significantly reduced number of parameters. However, the efficiency gains of ALBERT.

[5] Title: Universal Language Model Fine-tuning for Text Classification (ULMFiT)

Author: Howard, J., & Ruder.S

ULMFiT (Universal Language Model Fine-tuning for Text Classification) was one of the early and influential works in transfer learning for NLP. It proposed a three-stage approach: first, pre-training a general-purpose language model (AWD-LSTM) on a large corpus; second, fine-tuning this language model on the target task's data using techniques like discriminative fine-tuning (using different learning rates for different layers) and slanted triangular learning rates; and third, fine-tuning the classification layers. ULMFiT demonstrated that a pre-trained language model could be effectively adapted to various text classification tasks, especially with limited labeled data. While highly successful, its reliance on LSTM architectures means it typically has longer training times compared to transformer-based models, and LSTMs are now generally considered less efficient for large-scale NLP tasks.

[6] Title: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (T5)

Author: Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena .M

T5 (Text-to-Text Transfer Transformer) introduced a novel perspective by reframing all NLP tasks, including classification, question answering, and summarization, into a text generation problem. This was achieved by pre-training a massive transformer model on a new, large dataset called the Colossal Clean Crawled Corpus (C4). The model takes text as input and produces text as output for all tasks. This unified approach the fine-tuning process, as model and training objective could be used for diverse tasks. T5 achieved state-of-the-art results on many benchmarks. However, its large size makes it computationally expensive to train and deploy, especially in resource-constrained environments.

[7] Title: BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

Author: Lewis, M., Liu, Y., Goyal, N., Ghazvininejad. M

BART (Bidirectional and Auto-Regressive Transformer) is a sequence-to-sequence model that combines the strengths of BERT (bidirectional encoder) and GPT (autoregressive decoder) through a denoising autoencoder framework. During pre-training, BART corrupts input text in various ways (e.g., masking tokens, deleting tokens, shuffling sentences) and then trains the model to reconstruct the original text. This pre-training objective allows BART to learn effective representations for both understanding and generating text. It has shown strong performance on a wide range of tasks, including summarization, text generation, and question answering, particularly in low-data scenarios. Despite its effectiveness, BART is computationally intensive and can struggle with very long input sequences.

[8] Title: BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

Author: Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed. A

This is the same paper as [7]. As mentioned above, BART is a denoising autoencoder that excels in sequence-to-sequence tasks by learning to reconstruct original text from corrupted versions. Its pre-training approach integrates bidirectional encoding (like BERT) and autoregressive decoding (like GPT). This makes it particularly effective for tasks like text generation, summarization, and question answering, especially when the amount of task-specific training data is limited. However, like other large transformer models, it requires significant computational resources and can face challenges with very long documents.

[9] Title: DeBERTa: Decoding-enhanced BERT with Disentangled Attention

Author: He, P., Liu, X., Gao, J., & Chen, W. (2020).

BERTa (Decoding-enhanced BERT with Disentangled Attention) is an improvement over the original BERT architecture. It introduces disentangled attention, where each word's content and position are represented by two separate vectors, and the attention weights between words are computed using these disentangled matrices. This allows the model to better understand the roles of content and position in a sequence. DeBERTa also incorporates an enhanced mask decoder for the masked language modeling task and a new virtual adversarial training method for fine-tuning. These innovations led to significant performance improvements on several NLP benchmarks, even surpassing human performance on the SuperGLUE benchmark in some cases. However, achieving optimal performance with DeBERTa often requires careful fine-tuning strategies due to its complexity and large size.

[10] Title: Efficient Transformers: A Survey

Author: Tay, Y., Dehghani, M., Bahri, D., & Metzler, D. Efficient transformers.

This paper provides a comprehensive overview of various techniques developed to improve the efficiency of Transformer models. The increasing size and computational cost of transformers pose challenges for training and deployment, especially for long sequences. The survey categorizes these efficiency strategies into four main areas: low-rank factorization (reducing the dimensionality of weight matrices), kernel-based attention approximations (using more efficient ways to compute attention), memory-efficient architectures (designing models that require less memory), and sparse attention mechanisms (focusing attention on the most relevant parts of the input).

**CHAPTER 3**

**SYSTEM REQUIREMENTS**

### 3.1 HARDWARE REQUIREMENTS

* CPU: Intel Core i5 or better
* GPU: NVIDIA GTX 1080 or higher
* Hard Disk: 256GB SSD
* RAM: 8GB or more
* Optional; NVIDIA GPU (CUDA compatible) recommended for accelerating embedding training with deep learning models

### 3.2 SOFTWARE REQUIREMENTS

* Windows 10/11, Ubuntu 20.04+, or macOS
* Machine Learning Framework: PyTorch (v1.7+) or TensorFlow (v2.5+)
* Image Processing Library: OpenCV (v4.5+)
* IDE: Visual Studio Cod (v1.60+) or Jupyter Notebook (v6.0+)
* Operating System: Windows 10 or higher
* Optional Data Analysis Tools: Pandas (v1.1+) and Matplotlib (v3.3+) for data visualization

**CHAPTER 4**

**SYSTEM OVERVIEW**

**4.1 EXISTING SYSTEM**

The existing systems for word embedding have primarily been developed and optimized for high-resource languages like English, where large-scale annotated datasets and advanced linguistic tools are readily available. Models such as Word2Vec, GloVe, FastText, and BERT have demonstrated remarkable performance in representing word semantics and context. However, their effectiveness significantly drops when applied directly to low-resource languages like Tamil due to the lack of sufficient data, limited linguistic resources, and the unique morphological richness of the Tamil language. In the case of Tamil, researchers have mostly relied on pre-trained FastText models, which provide subword-level embeddings, offering moderate improvements in handling Tamil’s complex grammar.

**4.1.1 DRAWBACKS OF EXISTING SYSTEM**

Despite notable progress in word embedding techniques for high-resource languages, the existing systems present several limitations when applied to the Tamil language. Firstly, most embedding models like Word2Vec, GloVe, and even BERT are trained primarily on English corpora and fail to capture the linguistic intricacies of Tamil, which is a morphologically rich and agglutinative language. The limited availability of large, clean, and domain-specific Tamil corpora further hampers the performance of these models. Secondly, although models like FastText address subword information and perform better on Tamil compared to others, they still lack deeper contextual understanding.

**4.2 PROPOSED SYSTEM**

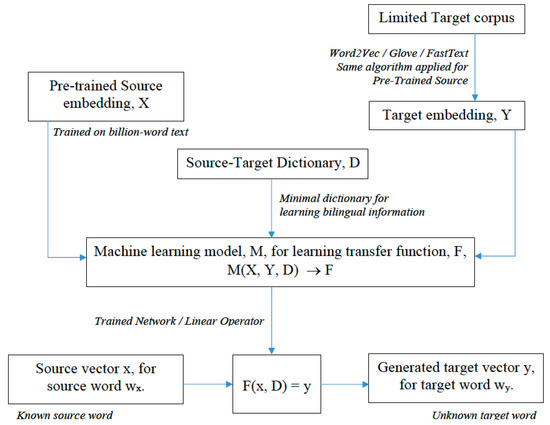
The proposed system aims to develop a dedicated word embedding framework specifically tailored for the Tamil language, addressing the limitations of existing models. This system will utilize advanced embedding techniques such as FastText or Word2Vec, trained on a large, cleaned Tamil corpus to effectively capture the rich morphological and semantic features of the language. To enhance the model’s accuracy, the system incorporate subword information, thereby improving the representation of inflected and compound Tamil words. In addition to generating high-quality embeddings system will include an interactive visualization module that allows users to explore word relationships in 2D or 3D space using dimensionality reduction techniques like PCA or t-SNE.

**4.2.1 ADVANTAGES OF PROPOSED SYSTEM**

The proposed system offers several advantages over existing approaches. Firstly, it is specifically designed for the Tamil language, addressing its unique grammatical structure and morphological complexity, which are often overlooked in general-purpose models. By using subword-level embedding techniques such as FastText or Word2Vec, the system ensures better representation of Tamil words, including rare or compound terms. Secondly, it includes an intuitive visualization component that enables users to explore the relationships between words in a clear and interactive way, enhancing interpretability and usability. This makes the system especially beneficial for linguists, educators, and students who need to analyze semantic patterns.

**CHAPTER 5**

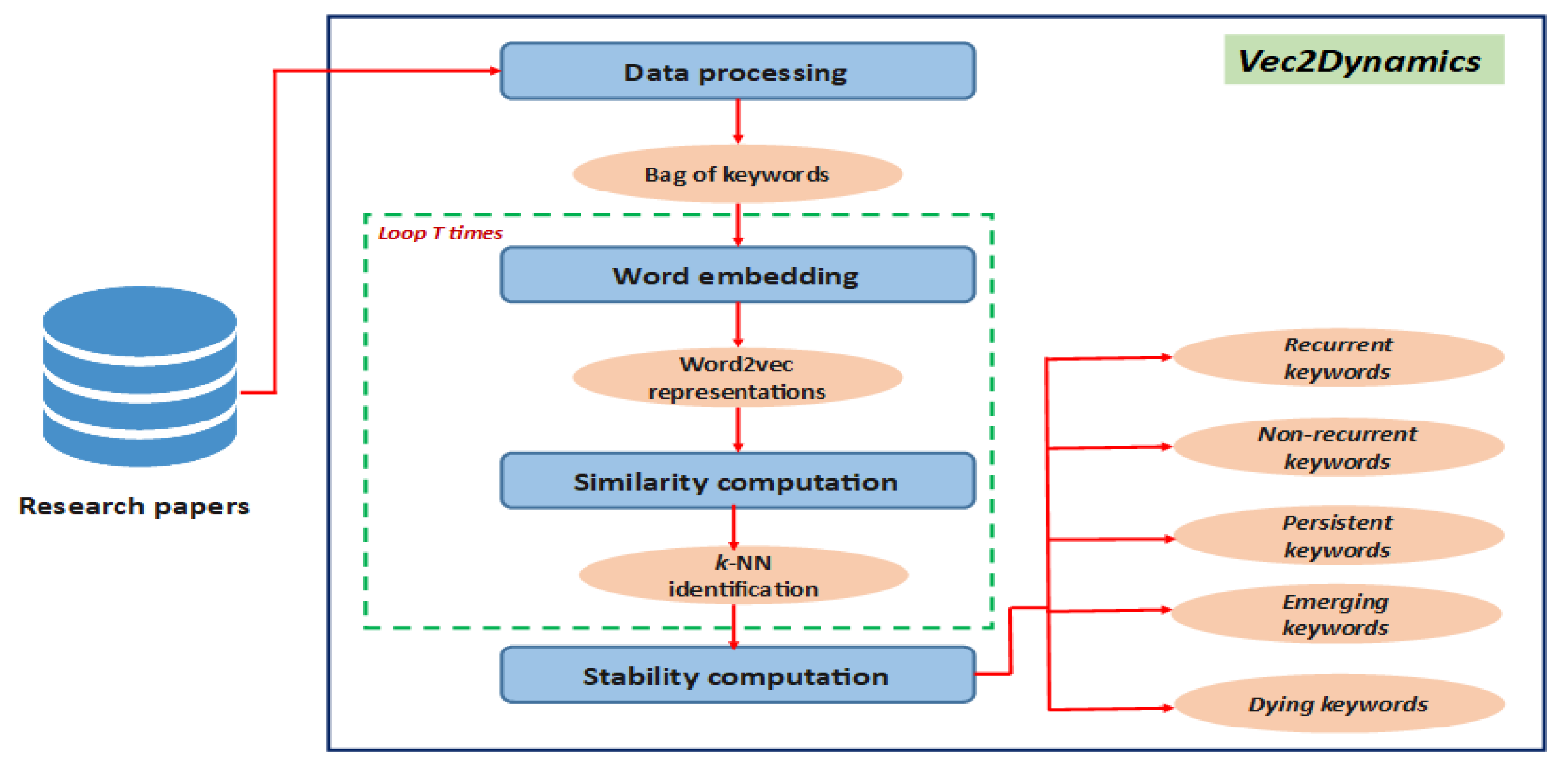
**SYSTEM IMPLEMENTATION**

****

# Fig 5.1 *Overall system diagram for word Embedding*

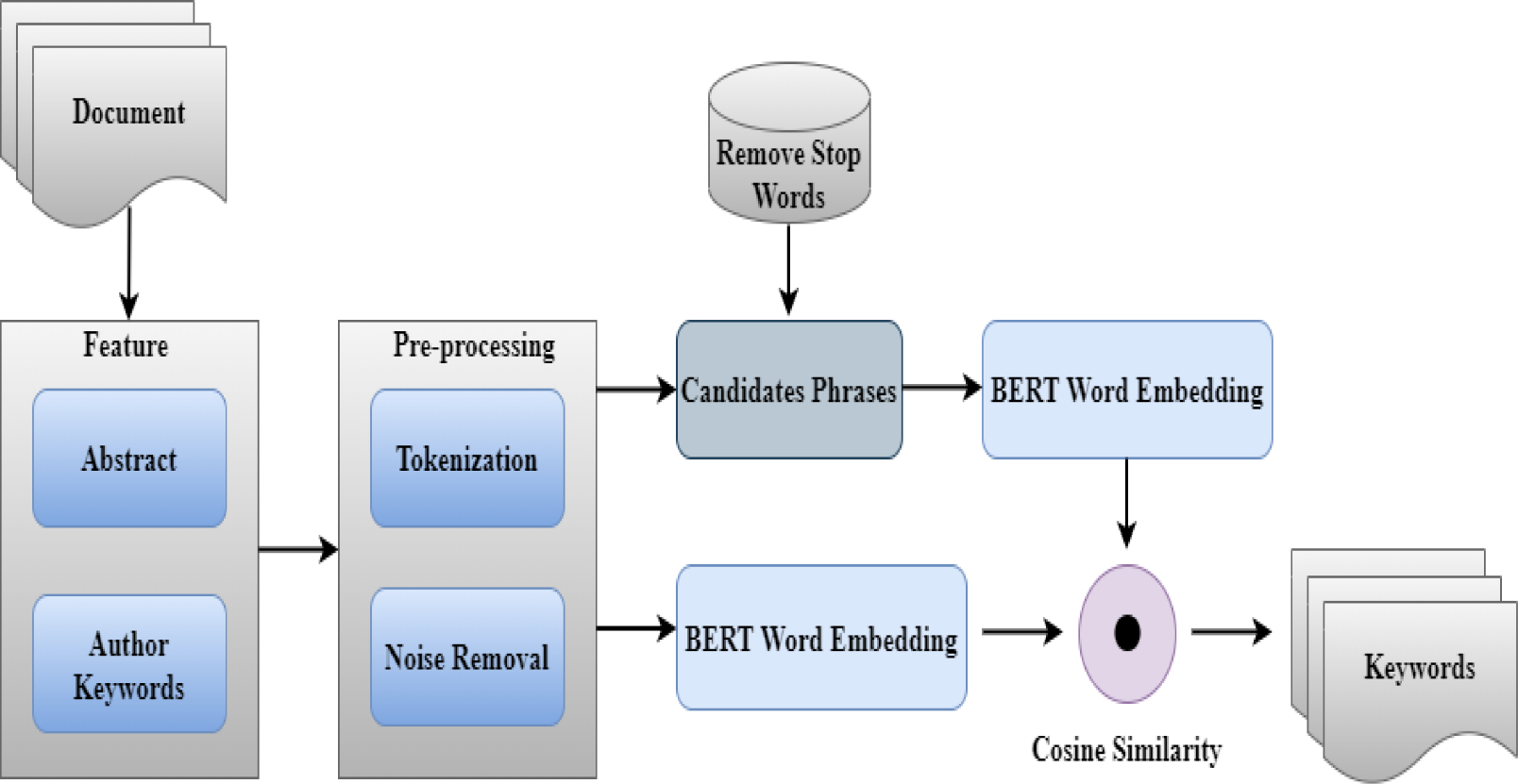
# 

**5.1 SYSTEM ARCHIECTURE**



# 5.2 SYSTEM FLOW

The system begins by collecting Tamil language documents that serve as the primary input. From these documents, important features such as abstracts and author-provided keywords are extracted to focus on the relevant textual content. Once the features are obtained, the text undergoes a pre-processing stage, which includes tokenization and noise removal to clean and standardize the data. After preprocessing, candidate phrases are generated and filtered by removing stop words to ensure meaningful and context-rich terms are retained. These candidate phrases are then passed through a word embedding model—such as BERT or Word2Vec—which converts the words into high-dimensional vector representations.

****

***Fig 5.2*** *Overall System flow*

**5.3 LIST OF MODULES**

* Text Corpus Collection and Dataset Preparation Module
* Data Preprocessing Module
* Word Embedding Generation Module
* Dimensionality Reduction and Visualization Module
* Offensive Content Classification Module

**5.4 MODULE DESCRIPTION**

It captures both semantic and syntactic similarities between words, making it useful for clustering and similarity-based visualizations. However, given Tamil’s rich morphological structure, FastText is more suitable, as it improves upon Word2Vec by representing each word as a combination of character n-grams

**5.4.1 TEXT CORPUS COLLECTION AND DATASET PREPARATION MODULE**

This module focuses on collecting a large set of Tamil text data, which may include tweets, social media posts, movie reviews, or any domain-specific Tamil content. The gathered data is then annotated or categorized based on the presence of offensive or non-offensive content (if applicable). This step ensures that the dataset is representative and suitable for training and evaluation.

### 5.4.2 DATA PREPROCESSING MODULE

### In this module, the raw Tamil text is cleaned and normalized. The preprocessing steps may include removing special characters, stop words, extra white spaces, and correcting inconsistent spelling or encoding issues. Tokenization is also performed to split text into meaningful units (words or subwords), preparing it for embedding. For Tamil, language-specific preprocessing techniques may be applied to preserve linguistic accuracy.

### 5.4.3 WORD EMBEDDING GENERATION MODULE

This module converts the processed Tamil tokens into dense vector representations using word embedding techniques like Word2Vec, FastText, or multilingual BERT. These embeddings capture semantic relationships between Tamil words and are crucial for further analysis or classification. Pretrained or custom-trained embeddings may be used depending on the dataset and objective.

### 5.4.4 DIMENSIONALITY REDUCTION AND VISUALIZATION MODULE

To visualize the high-dimensional word vectors, dimensionality reduction techniques like t-SNE or PCA are applied. This module helps project the Tamil word embeddings into a 2D or 3D space for easier interpretation. The resulting visualizations allow users to observe word clusters, semantic groupings, and contextual patterns, particularly focusing on offensive versus non-offensive word distributions.

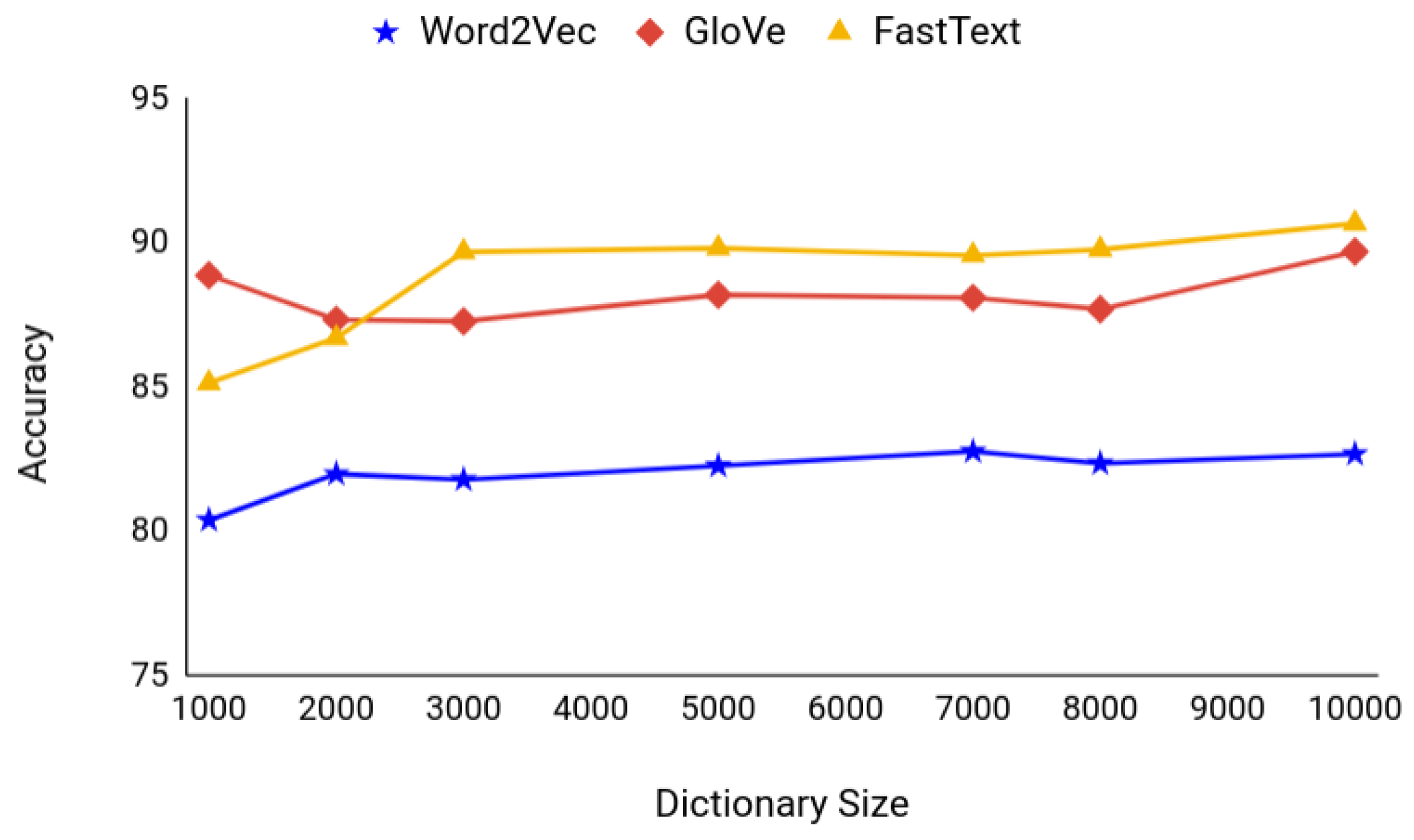
### 5.4.5 OFFENSIVE CONTENT CLASSIFICATION MODULE

This module utilizes the embedded word vectors to classify content as offensive or non-offensive. A machine learning or deep learning model is trained on the labeled embeddings to detect hate speech, vulgar terms, or toxic language in Tamil. The output of this module can help moderate content or improve natural language understanding for regional languages. These models reduce the vector dimensions for visualization purposes while preserving the relationships between word meanings, enabling better insights into how offensive and non-offensive terms are distributed in Tamil text data.

**CHAPTER-6**

**RESULT AND DISCUSSION**

The data set we use is a collection of political news articles of India from various newspapers in Tamil language. Our data set is huge and it has around 2.7 lakh sentences which contains 50 lakh words. So by using this huge data set or corpus which covers words of all classes (open and closed) like noun, verb, adjective, preposition, determiners etc., we will be able to draw a conclusion which gives an insight of the best of both models. Four words (Jayalalitha, Sethu, District and Chennai) in Tamil language are taken from the corpus and word similarity is found using the above mentioned 8 models.



***Fig 6.1*** *Performance Metrics*

### Inference:

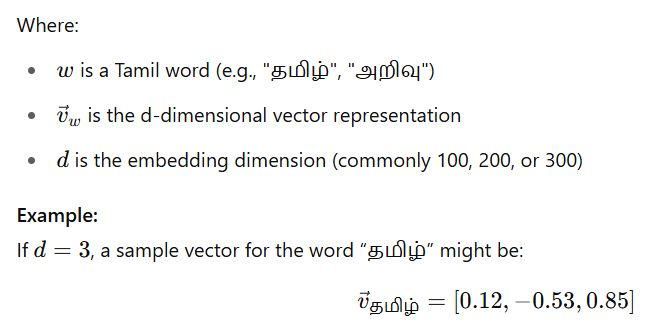
Word2Vec, GloVe, and FastText, across varying dictionary sizes. The x-axis represents the dictionary size, ranging from 1000 to 10000 unique words, while the y-axis indicates the accuracy achieved by each model, spanning from 75% to 95%. The blue line with star markers illustrates the accuracy of Word2Vec, starting around 80% and showing a modest increase before plateauing. The red line with diamond markers depicts GloVe's performance, beginning higher than Word2Vec, experiencing a slight dip, and then generally improving with larger vocabularies, eventually reaching approximately 90%.

### Mathematical Calculations:

**1. Vector Representation of Tamil Words (Word Embedding)**

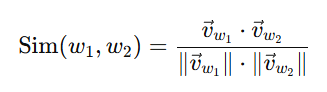
Each word in Tamil is converted into a dense numerical vector using models like Word2Vec or FastText. The word embedding is defined as:

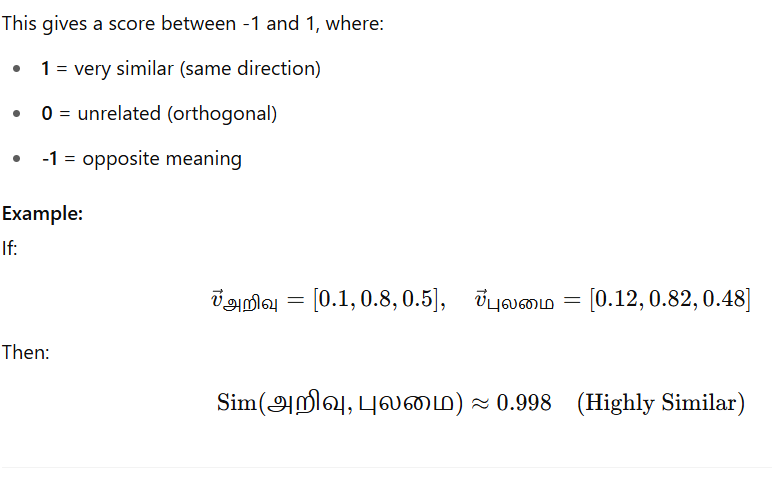




#### 2. Cosine Similarity for Semantic Similarity

To measure similarity between two Tamil words w1&w2 and w1&w2​, the cosine similarity is computed as:

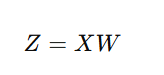


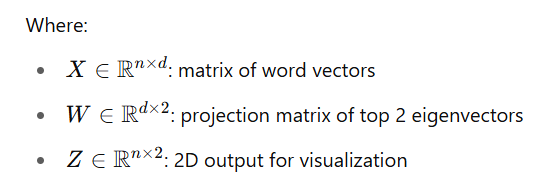


**3. Dimensionality Reduction for Visualization**

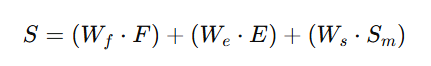
To project high-dimensional vectors (e.g., 100D) to 2D space for plotting, Principal Component Analysis (PCA) or t-SNE is used.

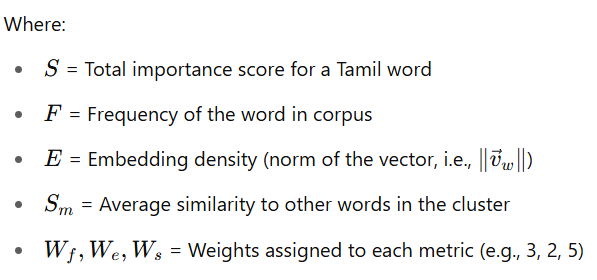
**PCA Formula:**

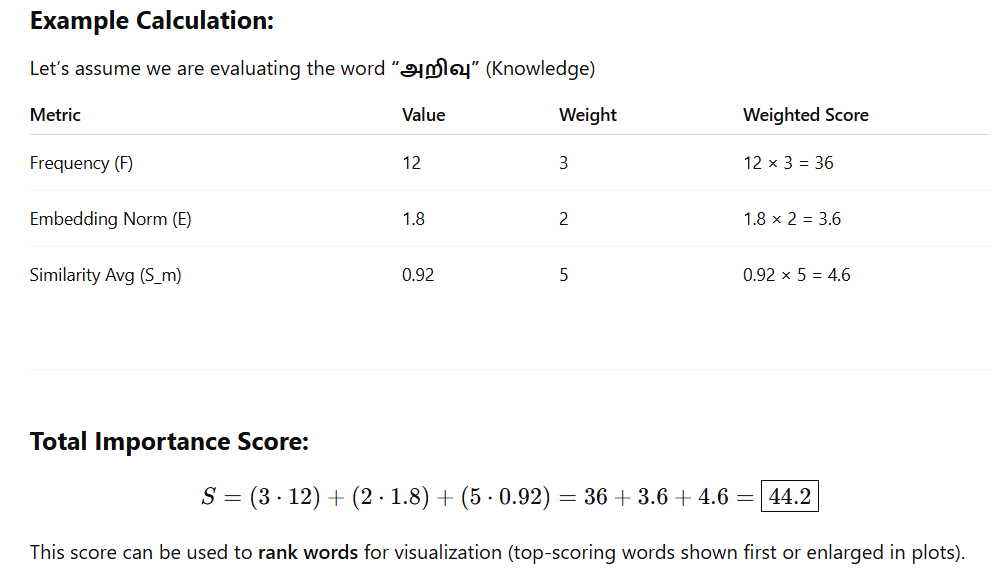




**Model Calculation Example:**





****

**APPENDIX**

**SAMPLE CODE**

# Down\_grade numpy if you want to strictly match

!pip install numpy==1.24.4

!pip install --upgrade gensim

!pip install git+https://github.com/facebookresearch/fastText.git

from google.colab import files

files.upload()

import os

os.makedirs('/root/.kaggle', exist\_ok=True)

os.rename('kaggle.json', '/root/.kaggle/kaggle.json')

os.chmod('/root/.kaggle/kaggle.json', 600)

!kaggle datasets download -d manidaaw/tamil-oscar-corpus

import zipfile

with zipfile.ZipFile("tamil-oscar-corpus.zip", 'r') as zip\_ref:

zip\_ref.extractall("tamil\_oscar")

def read\_corpus(path):

with open(path, 'r', encoding='utf-8') as f:

for line in f:

line = line.strip()

if line:

yield line.split()

corpus\_path = "/content/tamil\_oscar/ta\_part\_3.txt.gz"

sentences = list(read\_corpus(corpus\_path))

print("Number of sentences loaded:", len(sentences))

print("Sample:", sentences[0])

from gensim.models import Word2Vec

# Use a smaller subset for quicker training (adjust as needed)

short\_sentences = sentences[:50000]  # Only use first 10,000 lines

# Train with faster settings

w2v\_model = Word2Vec(

sentences=short\_sentences,

vector\_size=50,     # Smaller embedding size

window=5,           # Context window

min\_count=10,       # Ignore rare words

workers=4,          # Parallel training

sg=0,               # Use CBOW (faster than skip-gram)

epochs=5            # Fewer training iterations

)

# Save the model

w2v\_model.save("tamil\_word2vec\_quick.model")

print("Quick Word2Vec model trained and saved!")

# Load the model (optional if you're continuing in the same session)

from gensim.models import Word2Vec

model = Word2Vec.load("tamil\_word2vec\_quick.model")

# Get vector for a word

print(" Vector for தமிழ்:", model.wv["தமிழ்"][:10])  # First 10 dimensions

# Find most similar words

similar\_words = model.wv.most\_similar("தமிழ்", topn=5)

print("Similar words to 'தமிழ்':")

for word, score in similar\_words:

print(f"{word} - {score:.4f}")

# Input a Tamil sentence from the user

input\_sentence = input("Enter a Tamil sentence: ")

# Tokenize the sentence (basic whitespace-based split)

words = input\_sentence.strip().split()

print("\n Word analysis:")

for word in words:

if word in model.wv:

print(f"\nWord: {word}")

print(" Vector (first 5 dims):", model.wv[word][:5])

similar\_words = model.wv.most\_similar(word, topn=3)

print(" Top 3 similar words:")

for similar\_word, score in similar\_words:

print(f"{similar\_word} ({score:.4f})")

else:

print(f"\n Word '{word}' not found in vocabulary.")

!pip install gradio

!pip install matplotlib

# Install Tamil font

!apt-get -y install fonts-noto

import matplotlib.pyplot as plt

plt.rcParams['font.family'] = 'Noto Sans Tamil'

plt.rcParams['axes.unicode\_minus'] = False

import matplotlib.pyplot as plt

def plot\_similar\_words(word):

if word not in model.wv:

return f"'{word}' not found in vocabulary.", None

similar = model.wv.most\_similar(word, topn=5)

labels, scores = zip(\*similar)

# Plot

plt.figure(figsize=(6, 4))

plt.barh(labels, scores)

plt.xlabel("Similarity Score")

plt.title(f"Top 5 Similar Words to '{word}'")

plt.gca().invert\_yaxis()

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

import numpy as np

# Sample intent dataset

intent\_data = [

**("வணக்கம், எப்படி இருக்கிறீர்கள்?", "வாழ்த்து"),**

**("டிக்கெட் பதிவு செய்யவேண்டும்", "டிக்கெட் பதிவு"),**

**("ஒரு படம் பார்க்கவேண்டும்", "பொழுது வினோதம்"),**

**("எனது டிக்கெட்டை ரத்து செய்யவும்", "டிக்கெட் ரத்து"),**

**("நன்றி உங்கள் உதவிக்கு", "நன்றி"),**

**("மாலை ஷோக்கு டிக்கெட் வேண்டும்", "டிக்கெட் பதிவு"),**

**("எனது டிக்கெட்டை முடக்கியுங்கள்", "டிக்கெட் ரத்து"),**

**("உங்கள் சேவைக்கு நன்றி", "நன்றி"),**

**("நான் படம் பார்க்க விரும்புகிறேன்", "பொழுது வினோதம்"),**

**("வணக்கம்", "வாழ்த்து") ]**

# Convert sentence to averaged vector

def get\_sentence\_vector(sentence):

words = sentence.strip().split()

vectors = [model.wv[word] for word in words if word in model.wv]

return np.mean(vectors, axis=0) if vectors else np.zeros(model.vector\_size)

# Prepare dataset

X = [get\_sentence\_vector(text) for text, label in intent\_data]

y = [label for \_, label in intent\_data]

le = LabelEncoder()

y\_encoded = le.fit\_transform(y)

clf = LogisticRegression(max\_iter=1000)

clf.fit(X, y\_encoded)

def predict\_intent(tamil\_sentence):

vec = get\_sentence\_vector(tamil\_sentence)

pred = clf.predict([vec])[0]

return f" Predicted Intent: \*\*{le.inverse\_transform([pred])[0]}\*\*"

import gradio as gr

intent\_ui = gr.Interface(

inputs=gr.Textbox(label="Enter a Tamil Sentence"),

outputs="markdown",

title=" Tamil Chatbot Intent Classifier",

description="Enter a Tamil sentence and this classifier will predict the chatbot's intent."

)

import gradio as gr

import matplotlib.pyplot as plt

from matplotlib import font\_manager as fm

# Load Tamil font

tamil\_font = fm.FontProperties(fname="NotoSansTamil-Regular.ttf")

plt.rcParams['axes.unicode\_minus'] = False

# --- Function 1: Word Analysis ---

def analyze\_tamil\_sentence(input\_sentence):

words = input\_sentence.strip().split()

output = ""

for word in words:

if word in model.wv:

vector = model.wv[word][:5]

similar = model.wv.most\_similar(word, topn=3)

output += f" Word: {word}\n"

output += f" Vector (first 5 dims): {vector}\n"

output += " Top 3 similar words:\n"

for sw, score in similar:

output += f"   {sw} ({score:.4f})\n"

output += "\n"

else:

output += f" Word '{word}' not found in vocabulary.\n\n"

return output

# --- Function 2: Tamil Bar Chart for Similar Words ---

def plot\_similar\_words(word):

if word not in model.wv:

return f" '{word}' not found in vocabulary.", None

similar = model.wv.most\_similar(word, topn=5)

labels, scores = zip(\*similar)

plt.figure(figsize=(6, 4))

bars = plt.barh(range(len(labels)), scores, color="skyblue")

plt.yticks(range(len(labels)), labels, fontproperties=tamil\_font)

plt.xlabel("Similarity Score", fontproperties=tamil\_font)

plt.title(f"'{word}'-க்கு ஒத்த சொற்கள்", fontproperties=tamil\_font)

plt.gca().invert\_yaxis()

for i, bar in enumerate(bars):

score = bar.get\_width()

plt.text(score + 0.01, bar.get\_y() + bar.get\_height() / 2,

f"{score:.2f}", va='center', ha='left',

fontsize=9, fontproperties=tamil\_font)

return f" '{word}'-க்கு ஒத்த சொற்கள்:", plt

# --- Dummy Intent Classifier (Placeholder) ---

def dummy\_intent(text):

return " Detected Intent: (placeholder)"

# --- Gradio Interfaces ---

interface1 = gr.Interface(

fn=analyze\_tamil\_sentence,

inputs=gr.Textbox(lines=2, placeholder="Enter Tamil sentence here..."),

outputs="text",

title=" Tamil Word Explorer",

description="Explore vector and similar words for each Tamil word."

)

interface2 = gr.Interface(

fn=plot\_similar\_words,

inputs=gr.Textbox(label="Enter a Tamil Word"),

outputs=["text", gr.Plot()],

title=" Similar Word Visualization",

description="View the top 5 most similar Tamil words using a bar chart."

)

intent\_ui = gr.Interface(

fn=dummy\_intent,

inputs="text",

outputs="text",

title="Chatbot Intent Classifier",

description="(Demo only) Replace with actual classifier using IndicBERT or mBERT"

)

# --- Combine UIs ---

final\_tabs = gr.TabbedInterface(

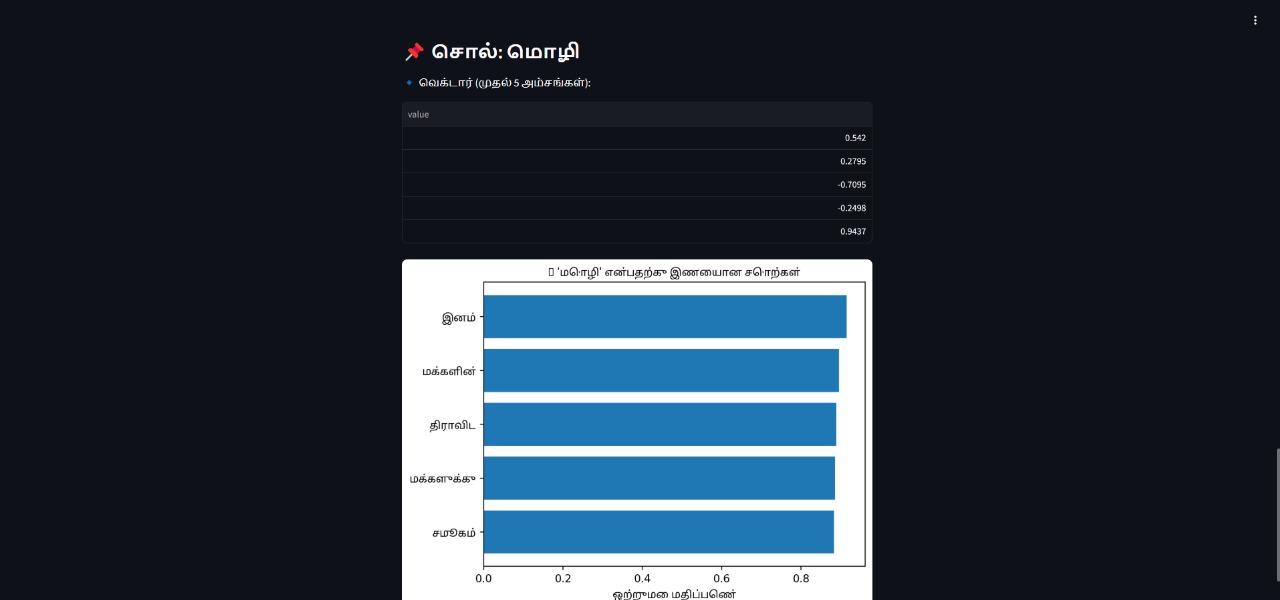
[interface1, interface2, intent\_ui],

["Word Explorer", "Similarity Plot", "Chatbot Intent Classifier"]

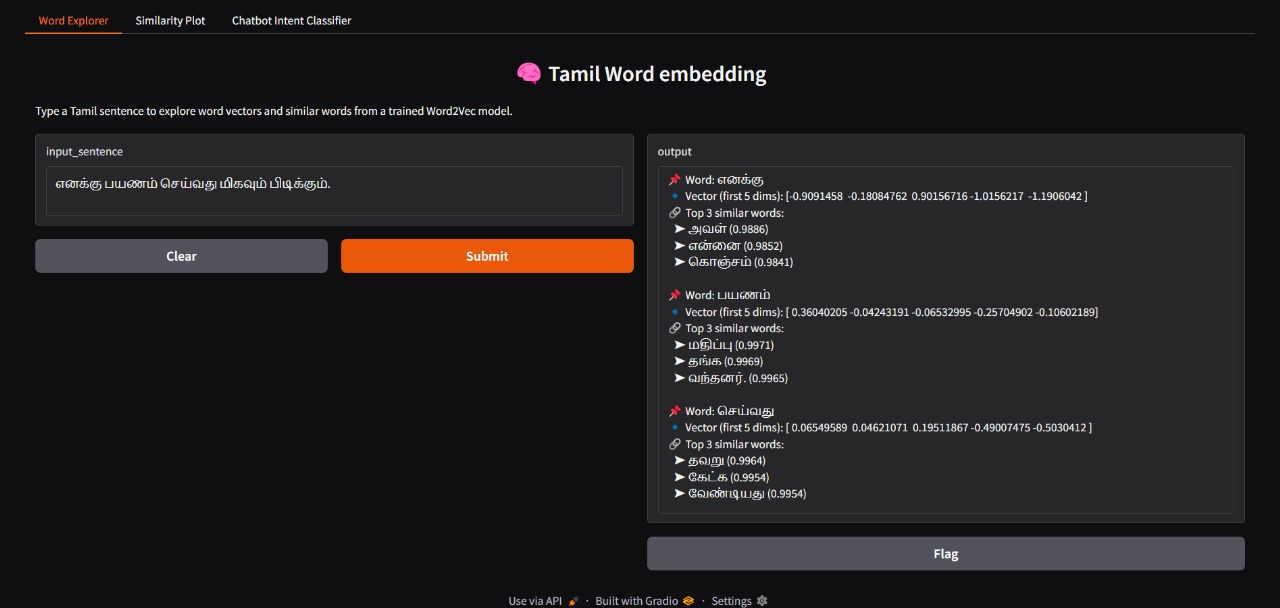
)

# Launch App

final\_tabs.launch(share=True)



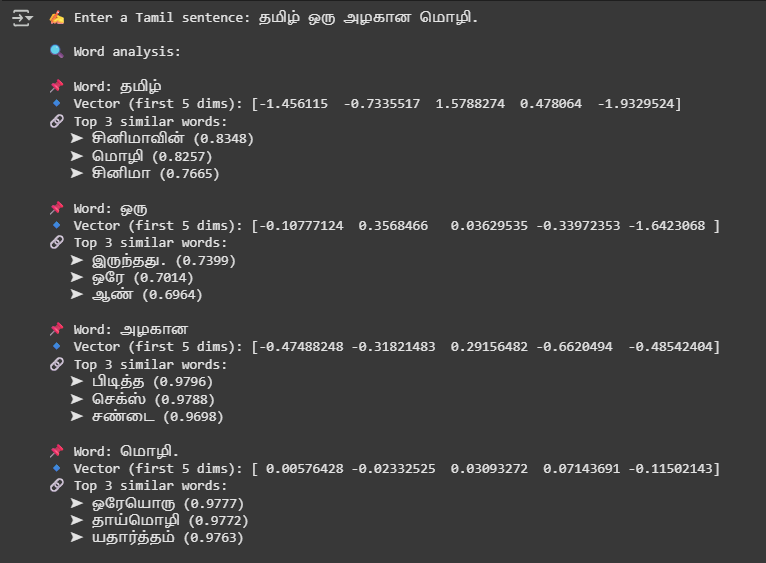
**OUTPUT SCREENSHOTS**

****

***Fig A.3*** *Tamil word embedding.*



***Fig A.4*** word embedding with chatbot



***Fig A.5*** *Tamil word embedding words in vector form*

**s**

**REFERENCE**

1. Torabi Asr, F.; Zinkov, R.; Jones, M. Querying Word Embeddings for Similarity and Relatedness.
2. Billah Nagoudi, E.M.; Ferrero, J.; Schwab, D.; Cherroun, H. Word Embedding-Based Approaches for Measuring Semantic Similarity of tamil Sentences.
3. Deerwester, S.; Dumais, S.T.; Furnas, G.W.; Landauer, T.K.; Harshman, R. Indexing by latent semantic analysis. *J.*
4. Fellbaum, C. *WordNet: An Electronic Lexical Database*; Language, Speech, and Communication; MIT Press: Cambridge, MA, USA, 1998.
5. Bengio, Y.; Ducharme, R.; Vincent, P.; Janvin, C. A Neural Probabilistic Language Model. *J. Mach. Learn*
6. Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; Kuksa, P.P. Natural Language Processing (almost) from Scratch.
7. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global Vectors for Word Representation.
8. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J. Distributed Representations of Words and Phrases and their Compositionality.
9. Wang, S.; Zhou, W.; Jiang, C. A survey of word embeddings based on deep learning.
10. Treviso, M.V.; Shulby, C.D.; Aluísio, S.M. Evaluating Word Embeddings for Sentence Boundary Detection in Speech Transcripts.