Download packages

```
!pip install gensim
!pip install torch torchtext

Collecting gensim
  Downloading gensim-4.3.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.1 kB)
Collecting numpy<2.0,>=1.18.5 (from gensim)
  Downloading numpy-1.26.4-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 2.6 MB/s eta
0:00:00
 gensim)
  Downloading scipy-1.13.1-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.6/60.6 kB 2.9 MB/s eta
0:00:00
ent already satisfied: smart-open>=1.8.1 in
/usr/local/lib/python3.12/dist-packages (from gensim) (7.3.1)
Requirement already satisfied: wrapt in
/usr/local/lib/python3.12/dist-packages (from smart-open>=1.8.1-
>gensim) (1.17.3)
Downloading gensim-4.3.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (26.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.6/26.6 MB 28.0 MB/s eta
0:00:00
py-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(18.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.0/18.0 MB 35.0 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (38.2 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.2/38.2 MB 14.3 MB/s eta
0:00:00
py, scipy, gensim
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: scipy
    Found existing installation: scipy 1.16.2
    Uninstalling scipy-1.16.2:
      Successfully uninstalled scipy-1.16.2
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2;
python_version >= "3.9", but you have numpy 1.26.4 which is
```

```
incompatible.
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2;
python_version >= "3.9", but you have numpy 1.26.4 which is
incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4
which is incompatible.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >=
"3.9", but you have numpy 1.26.4 which is incompatible.
tsfresh 0.21.1 requires scipy>=1.14.0; python_version >= "3.10", but
you have scipy 1.13.1 which is incompatible.
Successfully installed gensim-4.3.3 numpy-1.26.4 scipy-1.13.1
```

{"id":"0f75a1115e674065877dadfaac560907","pip_warning":{"packages":
["numpy"]}}

```
Requirement already satisfied: torch in
/usr/local/lib/python3.12/dist-packages (2.8.0+cu126)
Collecting torchtext
  Downloading torchtext-0.18.0-cp312-cp312-
manylinux1_x86_64.whl.metadata (7.9 kB)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from torch) (3.19.1)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in
/usr/local/lib/python3.12/dist-packages (from torch) (3.5)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
```

```
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in
/usr/local/lib/python3.12/dist-packages (from torch) (2.27.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.4.0 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.4.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-
packages (from torchtext) (4.67.1)
Requirement already satisfied: requests in
/usr/local/lib/python3.12/dist-packages (from torchtext) (2.32.4)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (from torchtext) (1.26.4)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch)
(1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests->torchtext)
(3.4.3)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests->torchtext)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests->torchtext)
(2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests->torchtext)
(2025.8.3)
Downloading torchtext-0.18.0-cp312-cp312-manylinux1_x86_64.whl (2.0
MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 26.0 MB/s eta
0:00:00
ost recent call last):
  File
"/usr/local/lib/python3.12/dist-packages/pip/_internal/cli/base_comman
d.py", line 179, in exc_logging_wrapper
^C
```

Load data

```python
import random
import numpy as np
import pandas as pd

from abc import ABC, abstractmethod

import spacy
from gensim.models import Word2Vec, FastText, KeyedVectors
import gensim.downloader as api

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.metrics.pairwise import cosine_similarity

import plotly.graph_objects as go
import plotly.express as px
from typing import List, Tuple, Optional, Union, Set

# Load dataset
# Link: https://www.kaggle.com/datasets/amontgomerie/cefr-levelled-
english-texts
df = pd.read_csv("/content/train.csv")
df = df.rename(columns={0: "label", 1:
"text"}).drop(columns=['label'])
print(df.head())
print(df.info())
```

```
                                                text
0  Hi!\nI've been meaning to write for ages and f...
1  It was not so much how hard people found the ...
2  Keith recently came back from a trip to Chicag...
3  The Griffith Observatory is a planetarium, and...
4  -LRB- The Hollywood Reporter -RRB- It's offici...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1494 entries, 0 to 1493
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    1494 non-null   object
dtypes: object(1)
memory usage: 11.8+ KB
None
```

Embedding strategies

```
class EmbeddingStrategy(ABC):
    """
    Abstract base class for all embedding strategies.
    Defines a standard interface for training and extracting word
vectors.
    """

    @abstractmethod
    def train(self, tokenized_sentences: List[List[str]]) ->
Union[Word2Vec, FastText, KeyedVectors]:
        """
        Train an embedding model on tokenized sentences.

        Args:
            tokenized_sentences (List[List[str]]): A list of
sentences, where each sentence is a list of tokens.

        Returns:
            Union[Word2Vec, FastText, KeyedVectors]: The trained or
loaded embedding model.
        """
        pass

    @abstractmethod
    def extract_vectors(
        self,
        model: Union[Word2Vec, FastText, KeyedVectors],
        vocab: Optional[Union[Set[str], List[str]]] = None
    ) -> List[Tuple[str, np.ndarray]]:
        """
        Extract word vectors from the trained model.

        Args:
            model (Union[Word2Vec, FastText, KeyedVectors]): The
trained or pre-trained embedding model.
            vocab (Optional[Union[Set[str], List[str]]]): Optional
vocabulary of words to extract. If None, extract all.

        Returns:
            List[Tuple[str, np.ndarray]]: List of tuples (word,
vector).
        """
        pass


class StrategyRegistry:
    """
    Registry to manage embedding strategies.
```

```python
    Allows registering and retrieving strategies by name.
    """
    _registry = {}

    @classmethod
    def register(cls, name: str):
        """
        Decorator to register a new embedding strategy.

        Args:
            name (str): Name of the strategy.
        """
        def decorator(strategy_cls):
            cls._registry[name.lower()] = strategy_cls
            return strategy_cls
        return decorator

    @classmethod
    def get_strategy(cls, name: str, **kwargs) -> EmbeddingStrategy:
        """
        Get an instance of a registered strategy by name.

        Args:
            name (str): Name of the registered strategy.
            **kwargs: Arguments to pass to the strategy constructor.

        Returns:
            EmbeddingStrategy: Instance of the selected strategy.

        Raises:
            ValueError: If the strategy is not registered.
        """
        name = name.lower()
        if name not in cls._registry:
            raise ValueError(f"Strategy '{name}' is not registered.")
        return cls._registry[name](**kwargs)

    @classmethod
    def list_strategies(cls) -> List[str]:
        """
        List all registered strategies.

        Returns:
            List[str]: List of strategy names.
        """
        return list(cls._registry.keys())


@StrategyRegistry.register("word2vec")
class Word2VecStrategy(EmbeddingStrategy):
    """
    Embedding strategy using gensim's Word2Vec.
    """

    def __init__(self, vector_size: int = 100, window: int = 5,
min_count: int = 1, workers: int = 4):
        self.vector_size = vector_size
        self.window = window
        self.min_count = min_count
        self.workers = workers

    def train(self, tokenized_sentences: List[List[str]]) -> Word2Vec:
        model = Word2Vec(
            sentences=tokenized_sentences,
            vector_size=self.vector_size,
            window=self.window,
            min_count=self.min_count,
            workers=self.workers
        )
        model.train(
            tokenized_sentences,
            total_examples=model.corpus_count,
            epochs=model.epochs
        )
        return model

    def extract_vectors(self, model: Word2Vec, vocab:
Optional[Union[Set[str], List[str]]] = None) -> List[Tuple[str,
np.ndarray]]:
        words = list(model.wv.index_to_key)
        if vocab is not None:
            words = [w for w in words if w in vocab]
        return [(word, model.wv[word]) for word in words]


@StrategyRegistry.register("fasttext")
class FastTextStrategy(EmbeddingStrategy):
    """
    Embedding strategy using gensim's FastText.
    """

    def __init__(self, vector_size: int = 100, window: int = 5,
min_count: int = 1, workers: int = 4):
        self.vector_size = vector_size
        self.window = window
        self.min_count = min_count
        self.workers = workers

    def train(self, tokenized_sentences: List[List[str]]) -> FastText:
        model = FastText(
            sentences=tokenized_sentences,
            vector_size=self.vector_size,
            window=self.window,
            min_count=self.min_count,
            workers=self.workers
        )
        model.train(
            tokenized_sentences,
            total_examples=model.corpus_count,
            epochs=model.epochs
        )
        return model

    def extract_vectors(self, model: FastText, vocab:
Optional[Union[Set[str], List[str]]] = None) -> List[Tuple[str,
np.ndarray]]:
        words = list(model.wv.index_to_key)
        if vocab is not None:
            words = [w for w in words if w in vocab]
        return [(word, model.wv[word]) for word in words]


@StrategyRegistry.register("glove")
class GloVeStrategy(EmbeddingStrategy):
    """
    Embedding strategy using pre-trained GloVe from gensim API.
    """

    def __init__(self, name: str = "glove-wiki-gigaword-100"):
        self.name = name

    def train(self, tokenized_sentences: List[List[str]]) ->
KeyedVectors:
        # Pre-trained model, ignore tokenized_sentences
        model = api.load(self.name)
        return model

    def extract_vectors(
        self,
        model: KeyedVectors,
        vocab: Optional[Union[Set[str], List[str]]] = None
    ) -> List[Tuple[str, np.ndarray]]:
        if vocab is None:
            words = list(model.key_to_index.keys())
        else:
            words = [w for w in vocab if w in model.key_to_index]
        return [(word, model[word]) for word in words]


class EmbeddingTrainer:
    """
    Wrapper class to train and extract word vectors using any
registered embedding strategy.
    """

    def __init__(self, strategy_name: str, **strategy_kwargs):
        self.strategy = StrategyRegistry.get_strategy(strategy_name,
**strategy_kwargs)
        self._nlp = spacy.load("en_core_web_sm", disable=["parser",
"ner", "tagger"])
        self._tokenized_sentences: Optional[List[List[str]]] = None

    def tokenize(self, text: str) -> List[str]:
        return [token.text.lower() for token in self._nlp(text) if
token.is_alpha]

    def tokenize_dataframe(self, df: pd.DataFrame, text_column: str) -
> List[List[str]]:
        return [self.tokenize(sentence) for sentence in
df[text_column]]

    def train(
        self,
        df: pd.DataFrame,
        text_column: str
    ) -> Union[Word2Vec, FastText, KeyedVectors]:
        self._tokenized_sentences = self.tokenize_dataframe(df,
text_column)
        return self.strategy.train(self._tokenized_sentences)

    def extract_word_vectors(
        self,
        model: Union[Word2Vec, FastText, KeyedVectors],
        df: Optional[pd.DataFrame] = None,
        text_column: Optional[str] = "text",
        vocab: Optional[Union[Set[str], List[str]]] = None
    ) -> List[Tuple[str, np.ndarray]]:
        if vocab is None:
            if self._tokenized_sentences is not None:
                vocab = {token for sentence in
self._tokenized_sentences for token in sentence}
            elif df is not None and text_column is not None:
                tokenized = self.tokenize_dataframe(df, text_column)
                vocab = {token for sentence in tokenized for token in
sentence}
            else:
                if hasattr(model, "wv"):
                    vocab = set(model.wv.index_to_key)
                else:
                    vocab = set(model.key_to_index.keys())
        return self.strategy.extract_vectors(model, vocab=vocab)
```

Utils

```
def reduce_dimensionality(
    vectors: np.ndarray,
    n_components: int = 3,
    random_state: int = 42
) -> np.ndarray:
    """
    Reduce the dimensionality of word vectors using PCA.

    Args:
        vectors (np.ndarray): Array (num_words, vector_dim).
        n_components (int): Target dimension (2 or 3).
        random_state (int): Random seed for reproducibility.

    Returns:
        np.ndarray: Reduced vectors (num_words, n_components).
    """
    if n_components not in (2, 3):
        raise ValueError("n_components must be 2 or 3.")
    if len(vectors) == 0:
        raise ValueError("Empty vector set provided for dimensionality
reduction.")

    reducer = PCA(n_components=n_components,
random_state=random_state)
    return reducer.fit_transform(vectors)


def visualize_embeddings(
    trainer,
    model,
    vocab: Optional[Union[set, list]] = None,
    dimension: int = 3,
    reduction_method: str = "pca"
):
    """
    Visualize word embeddings in 2D or 3D after PCA reduction.

    Args:
        trainer (EmbeddingTrainer): The embedding trainer used to
extract vectors.
        model: Trained or pre-trained embedding model.
        vocab (Union[set, list, None]): Optional subset of words to
visualize.
```

```
        dimension (int): 2 or 3.
        reduction_method (str): Only 'pca' is currently supported.
    """
    word_vectors = trainer.extract_word_vectors(model, vocab=vocab)
    if not word_vectors:
        print("⚠ No valid word vectors found for visualization.")
        return

    words = [word for word, _ in word_vectors]
    vectors = np.array([vec for _, vec in word_vectors])
    reduced_vectors = reduce_dimensionality(vectors,
n_components=dimension)

    if dimension == 3:
        fig = px.scatter_3d(
            x=reduced_vectors[:, 0],
            y=reduced_vectors[:, 1],
            z=reduced_vectors[:, 2],
            hover_name=words,
            title=f"3D PCA Visualization –
{trainer.strategy.__class__.__name__}"
        )
        fig.update_layout(scene=dict(
            xaxis_title='Component 1',
            yaxis_title='Component 2',
            zaxis_title='Component 3'
        ))
    elif dimension == 2:
        fig = px.scatter(
            x=reduced_vectors[:, 0],
            y=reduced_vectors[:, 1],
            hover_name=words,
            title=f"2D PCA Visualization –
{trainer.strategy.__class__.__name__}"
        )
        fig.update_layout(
            xaxis_title='Component 1',
            yaxis_title='Component 2'
        )
    else:
        raise ValueError("dimension must be 2 or 3")

    fig.update_traces(marker=dict(size=5))
    fig.show()

def top_k_similar_words(
    trainer,
    model,
    vocab: set,
    word: str,
```

```python
    k: int = 5
) -> List[str]:
    """
    Compute top-k words in vocab that are most similar to the given
word using cosine similarity.
    """
    word_vectors = trainer.extract_word_vectors(model, vocab=vocab)
    word_to_vec = {w: vec for w, vec in word_vectors}

    if word not in word_to_vec:
        raise ValueError(f"Word '{word}' not in model vocabulary or
provided vocab.")

    word_vec = word_to_vec[word].reshape(1, -1)
    other_words = [w for w in word_to_vec if w != word]
    other_vecs = np.array([word_to_vec[w] for w in other_words])

    sims = cosine_similarity(word_vec, other_vecs).flatten()
    top_idx = sims.argsort()[-k:][::-1]

    return [other_words[i] for i in top_idx]

def add_lines_to_origin(
    fig,
    reduced_vectors: np.ndarray,
    dimension: int = 3,
    line_color: str = 'black',
    line_width: int = 1
):
    """
    Add lines from origin to each point in reduced_vectors.
    """
    if dimension == 3:
        for xi, yi, zi in reduced_vectors:
            fig.add_trace(go.Scatter3d(
                x=[0, xi],
                y=[0, yi],
                z=[0, zi],
                mode='lines',
                line=dict(color=line_color, width=line_width),
                showlegend=False
            ))
    elif dimension == 2:
        for xi, yi in reduced_vectors:
            fig.add_trace(go.Scatter(
                x=[0, xi],
                y=[0, yi],
                mode='lines',
                line=dict(color=line_color, width=line_width),
                showlegend=False
            ))
    else:
        raise ValueError("dimension must be 2 or 3")

def visualize_top_k_words(
    trainer,
    model,
    vocab: set,
    k: int = 5,
    reduction_method: str = "pca",
    dimension: int = 2
):
    """
    Randomly select a word from vocab, find top-k similar words, and
visualize them
    with lines connecting to origin.
    """
    word = random.choice(list(vocab))
    print(f"� Randomly selected word: '{word}'")

    try:
        top_words = top_k_similar_words(trainer, model, vocab, word,
k=k)
    except ValueError as e:
        print(f"[WARN] {e}")
        return

    selected_words = [word] + top_words
    print(f"Top-{k} similar words: {top_words}")

    word_vectors = trainer.extract_word_vectors(model,
vocab=set(selected_words))
    if not word_vectors:
        print("⚠ No vectors found for selected words.")
        return

    words = [w for w, _ in word_vectors]
    vectors = np.array([vec for _, vec in word_vectors])
    reduced_vectors = reduce_dimensionality(vectors,
n_components=dimension)

    if dimension == 3:
        fig = go.Figure([
            go.Scatter3d(
                x=reduced_vectors[:, 0],
                y=reduced_vectors[:, 1],
                z=reduced_vectors[:, 2],
                mode='markers+text',
                marker=dict(size=5, color='blue'),
                text=words,
                textposition='top center'
            )
        ])
    else:
        fig = go.Figure([
            go.Scatter(
                x=reduced_vectors[:, 0],
                y=reduced_vectors[:, 1],
                mode='markers+text',
                marker=dict(size=5, color='blue'),
                text=words,
                textposition='top center'
            )
        ])

    add_lines_to_origin(fig, reduced_vectors, dimension=dimension)
    fig.update_layout(
        title=f"Top-{k} Words {dimension}D {reduction_method.upper()}
Visualization",
        scene=dict(
            xaxis_title='Component 1',
            yaxis_title='Component 2',
            zaxis_title='Component 3'
        ) if dimension == 3 else dict(
            xaxis_title='Component 1',
            yaxis_title='Component 2'
        )
    )
    fig.show()
```

Main

```
def run_all_strategies(
    df: pd.DataFrame,
    text_column: str,
    dimensions: List[int] = [2, 3],
    top_k: int = 5
):
    for strategy_name in StrategyRegistry.list_strategies():
        print(f"\n=== Running strategy: {strategy_name.upper()} ===")
        trainer = EmbeddingTrainer(strategy_name)
        model = trainer.train(df, text_column)
        print(f"Model trained/loaded successfully:
{type(model).__name__}")

        if trainer._tokenized_sentences is not None:
            vocab = {token for sentence in
trainer._tokenized_sentences for token in sentence}
```

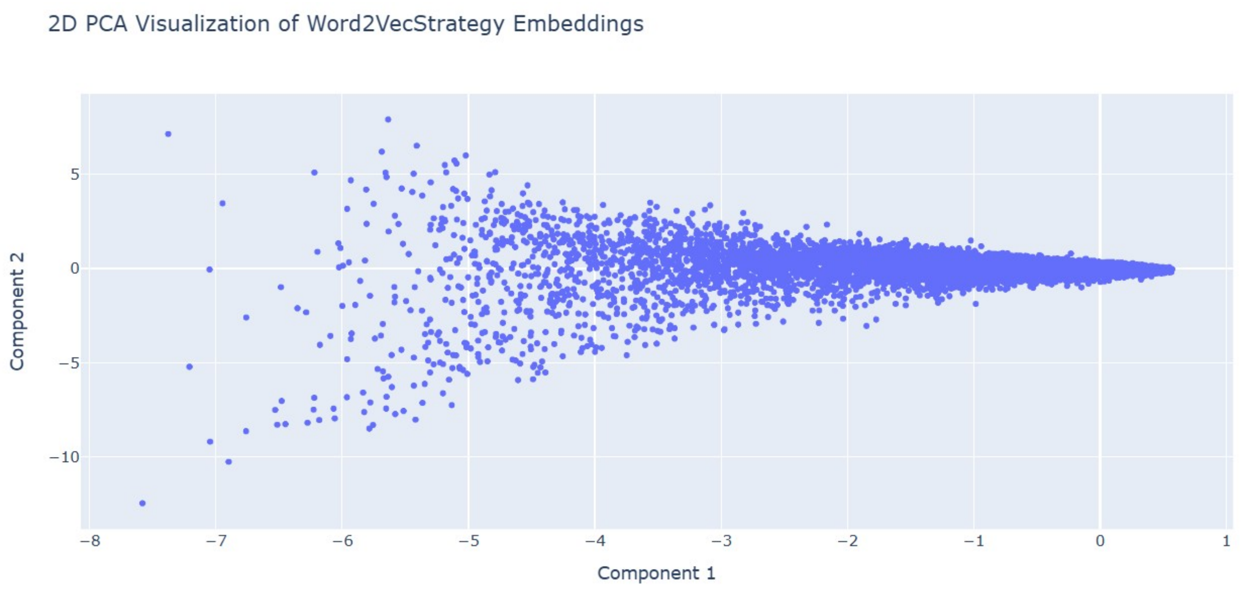
```
        else:
            vocab = set()
            for sentence in df[text_column]:
                tokens = trainer.tokenize(sentence)
                vocab.update(tokens)

        print(f"Vocab size: {len(vocab)}")

        for dim in dimensions:
            for reduction_method in ["pca"]:  # Chỉ dùng PCA, loại
bỏ t-SNE
                print(f"\nVisualizing {dim}D using
{reduction_method.upper()} for full vocab...")
                visualize_embeddings(
                    trainer=trainer,
                    model=model,
                    vocab=vocab,
                    dimension=dim,
                    reduction_method=reduction_method
                )

                print(f"Visualizing top-{top_k} similar words...")
                visualize_top_k_words(
                    trainer=trainer,
                    model=model,
                    vocab=vocab,
                    k=top_k,
                    reduction_method=reduction_method,
                    dimension=dim
                )
```

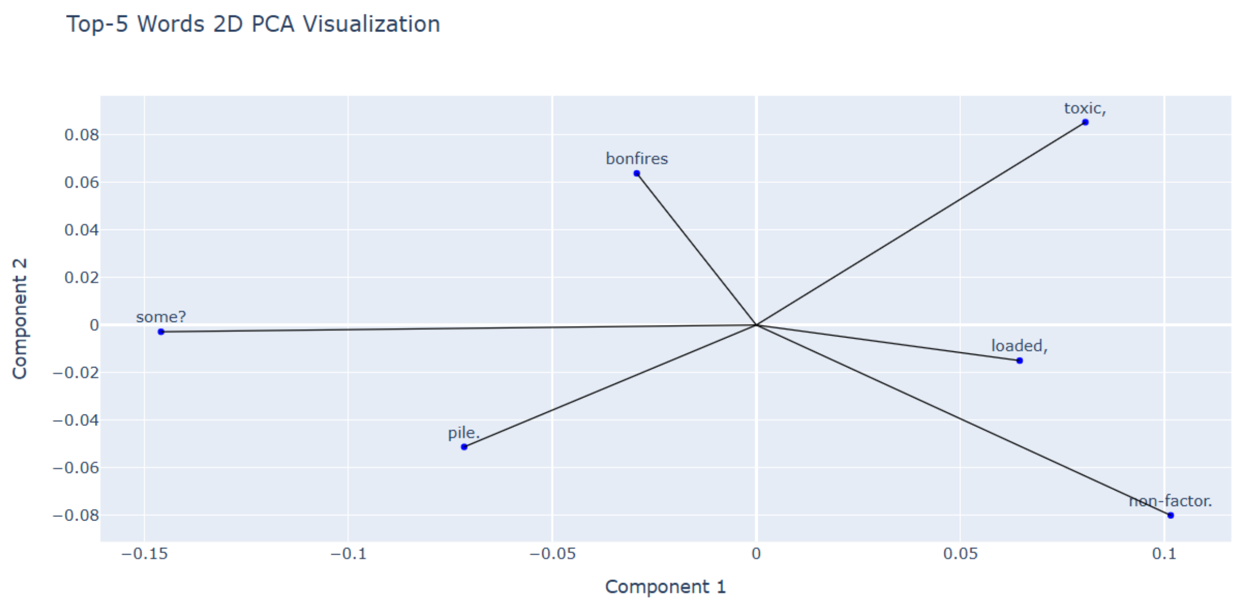
Test

```
run_all_strategies(df, "text")
```

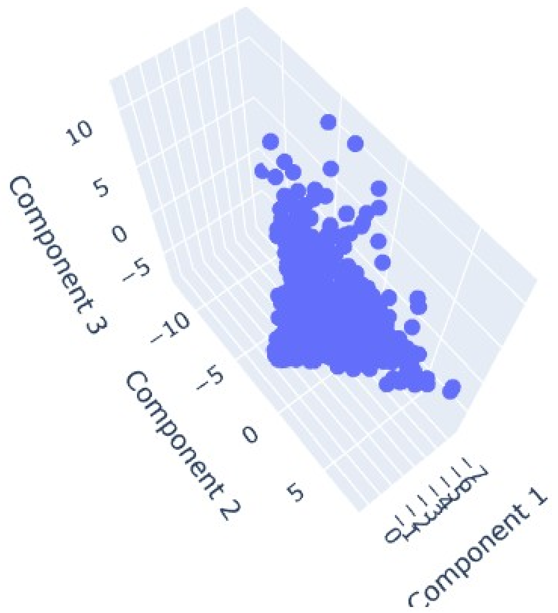
Result

W2V + PCA

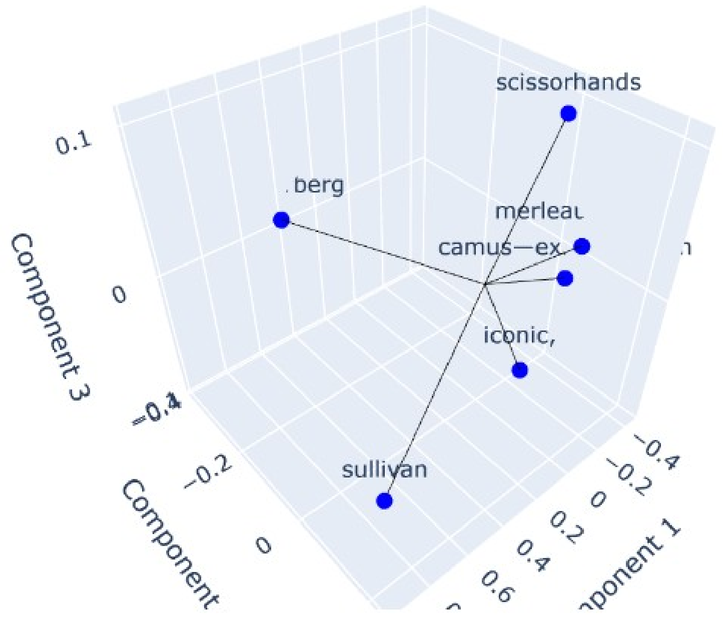




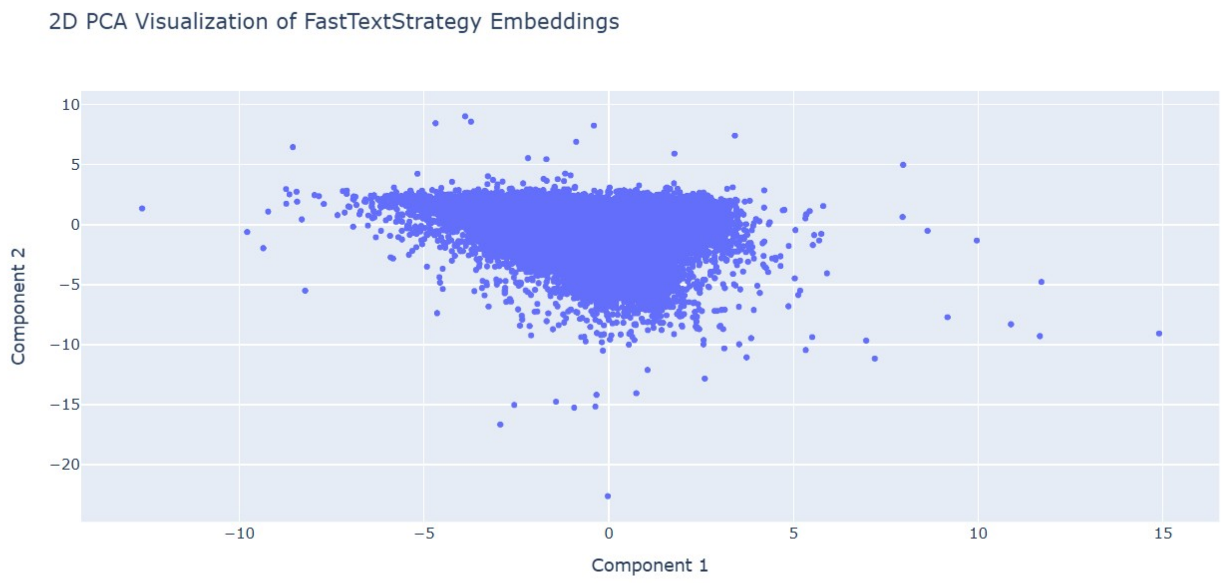
3D PCA Visualization of Word2VecStrategy Embeddings

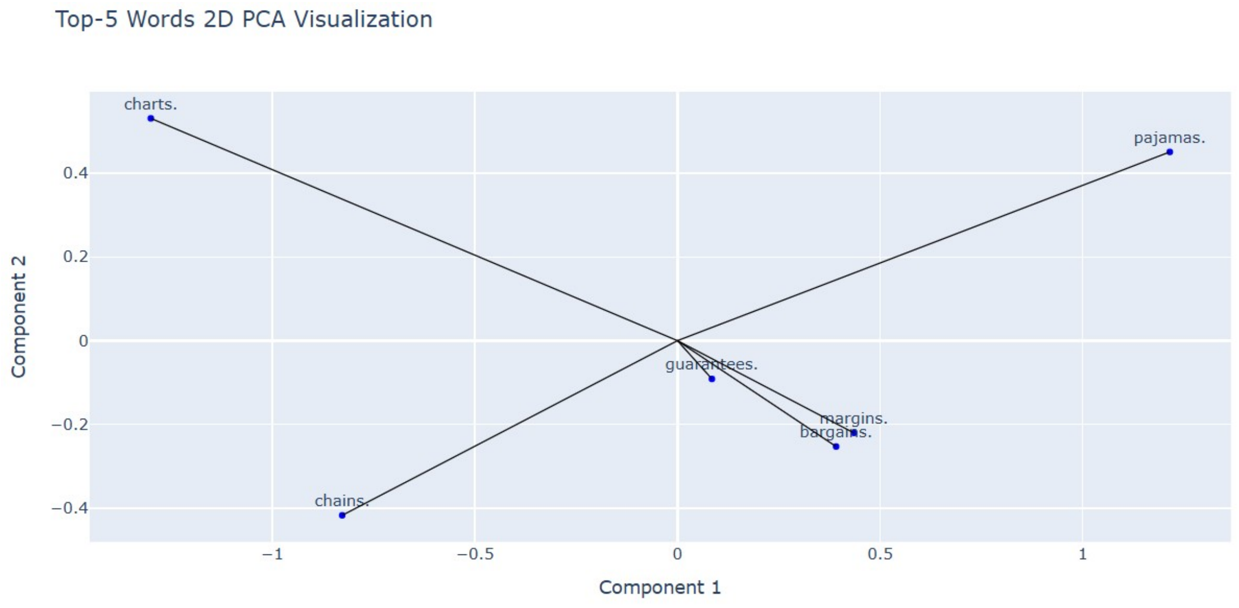


Top-5 Words 3D PCA Visualization

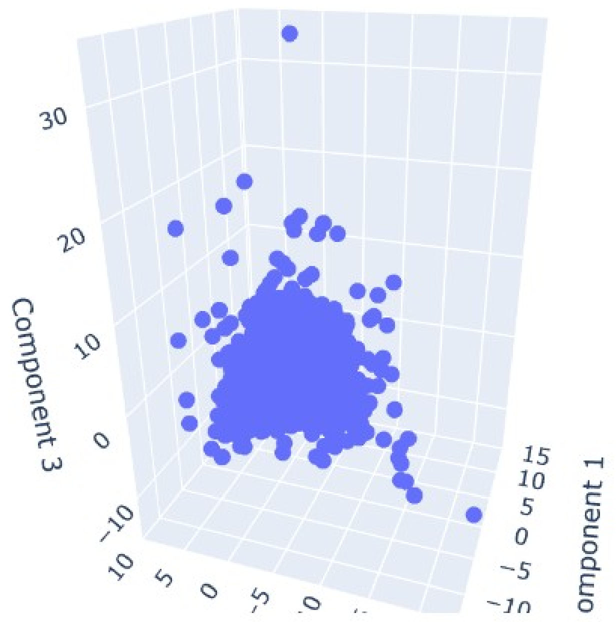


FastText + PCA

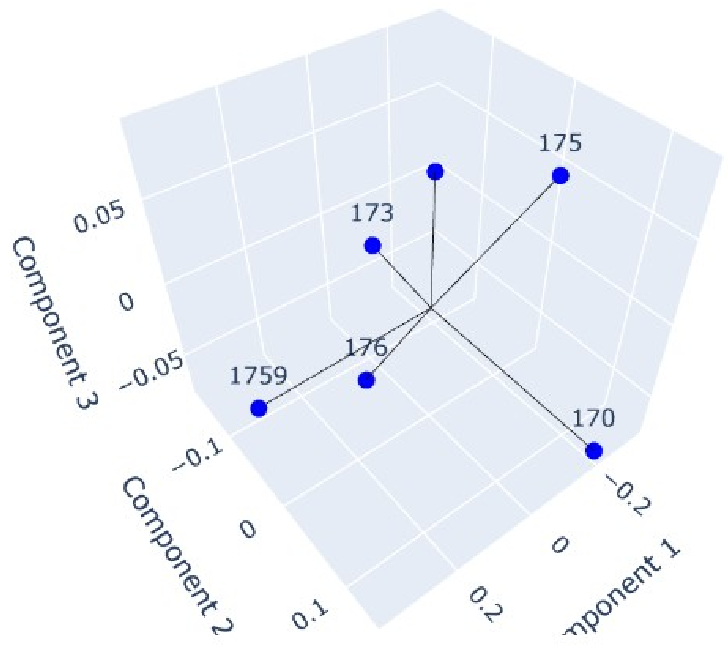




3D PCA Visualization of FastTextStrategy Embeddings



Top-5 Words 3D PCA Visualization



Nhận xét

Tổng quan ta thấy được cả 3 cách biểu diễn đều có thể có điểm mạnh và điểm yếu riêng tùy trường hợp.

1. Word2Vec học được quan hệ ngữ nghĩa và cú pháp tốt, tốc độ huấn luyện nhanh, tương thích tốt với nhiều ngôn ngữ.
2. Xử lý tốt từ hiếm, từ mới, và lỗi chính tả nhẹ, giữ được thông tin hình thái, hiệu quả cao trong bài toán ngôn ngữ nhỏ hoặc nhiễu.
3. Giữ được ngữ nghĩa toàn cục (global semantics) tốt hơn Word2Vec.