Natural Language Processing

ΚΑΝΔΗΛΑΣ ΑΝΔΡΕΑΣ Π22246

ΣΟΧΟΣ ΒΑΓΓΕΛΗΣ Π19242

Github Rep: https://github.com/andreaskan4/natural-language-proc.git

Introduction

Semantic reconstruction is a process in Natural Language Processing (NLP) that involves rewriting or restructuring text in order to improve its readability, fluency, or conciseness, while ensuring that the meaning of the original text is preserved. This task has become increasingly important as written communication often passes through several transformations—translations, paraphrases, summaries, or edits—before reaching the final audience.

In academic and professional contexts, semantic reconstruction can play a key role. For example, authors might need to reformulate passages of text so that they are more grammatically correct, clearer in meaning, or more stylistically appropriate for a given audience. Similarly, in fields such as healthcare or law, where precision of meaning is vital, reconstruction methods can help clarify poorly worded sentences without altering their intent.

In the present study, we worked with two Python scripts that tackled this challenge from complementary angles. The first file (pipelines.py) applied a series of linguistic and machine learning pipelines to generate alternative versions of the same input text using synonym replacement, paraphrasing, and summarization. The second file (paradoteo2.py) focused on evaluating and comparing the semantic similarity between original and reconstructed texts using modern embedding techniques such as Word2Vec, FastText, and BERT, alongside a simple custom baseline.

The purpose of this report is to document these approaches, explain their methodologies, analyze their results, and reflect on the strengths and weaknesses of different reconstruction techniques.

Background and Theoretical Foundations

Semantic Reconstruction in NLP

The task of rewriting text while preserving its meaning can be traced back to early work in rule-based natural language processing, where systems relied heavily on dictionaries, grammatical rules, and syntactic parsing. Over time, these methods were supplemented by statistical models that analyzed word co-occurrence in large corpora, such as n-gram models or vector space models.

With the introduction of distributed word representations like Word2Vec (Mikolov et al., 2013) and FastText (Bojanowski et al., 2017), NLP shifted towards embeddings that captured semantic meaning more effectively than sparse vector methods. Later, transformer-based models such as BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020) enabled contextualized embeddings and generative paraphrasing, vastly improving performance in tasks such as translation, summarization, and paraphrasing.

Embeddings:

Word2Vec: Uses a neural network to learn word vectors by predicting a word from its surrounding context (skip-gram) or predicting surrounding words from the current one (CBOW). Words that occur in similar contexts are placed closer in vector space.

FastText: Builds on Word2Vec by representing each word as a combination of character n-grams. This allows it to handle rare words or misspellings better, since even unseen words can be broken down into subword units.

BERT (Bidirectional Encoder Representations from Transformers): Unlike static embeddings, BERT generates contextual embeddings—meaning that the same word can have different vector representations depending on its sentence context. This is crucial for capturing nuanced meaning.

Bag-of-Words (BoW): A simple representation where each text is converted into a vector of word counts. While easy to compute, it ignores context and word order, making it less effective for capturing meaning.

Similarity and Dimensionality Reduction

Cosine Similarity: Measures the cosine of the angle between two vectors. For two embeddings AAA and BBB, the similarity is given by:

$\text{cosine}(A,B)=(A \cdot B)/(||A|| * ||B||)$

A similarity close to 1 means the vectors (and hence the texts) are very similar.

PCA (Principal Component Analysis): A statistical technique that reduces high-dimensional embeddings into lower dimensions (2D or 3D) while preserving as much variance as possible.

t-SNE (t-Distributed Stochastic Neighbor Embedding): A nonlinear dimensionality reduction technique that excels at visualizing clusters of semantically similar points.

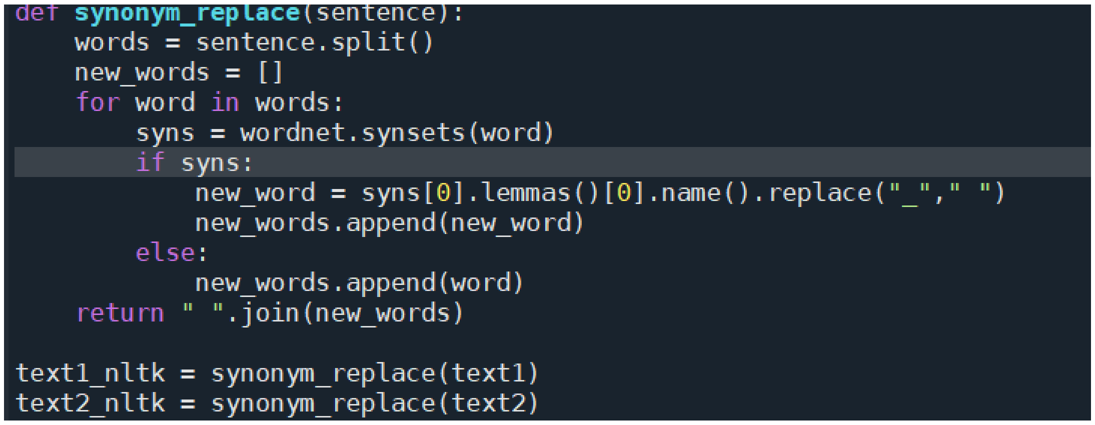
Methodology

Our work consisted of two main components, corresponding to the two Python scripts provided:

1. Pipelines for Text Reconstruction (pipelines.py)

The first script aimed to produce multiple reconstructions of two input texts. The main strategies used were:

a) Synonym Replacement with WordNet

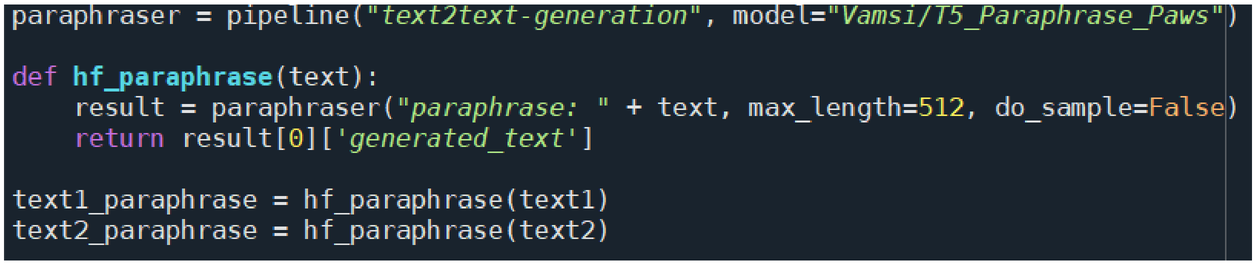


```python
def synonym_replace(sentence):
    words = sentence.split()
    new_words = []
    for word in words:
        syns = wordnet.synsets(word)
        if syns:
            new_word = syns[0].lemmas()[0].name().replace("_"," ")
            new_words.append(new_word)
        else:
            new_words.append(word)
    return " ".join(new_words)

text1_nltk = synonym_replace(text1)
text2_nltk = synonym_replace(text2)
```

We used the NLTK library's WordNet database to replace words with their synonyms. The function synonym_replace iterates over each word in the sentence, retrieves its synonyms from WordNet, and substitutes the word with the first available synonym. For example, "festival" might be replaced with "fête".

While straightforward, this method illustrates the limitations of rule-based NLP. Words can have multiple senses, and blindly replacing them with synonyms often leads to awkward or incorrect sentences.

b) Paraphrasing with Hugging Face T5

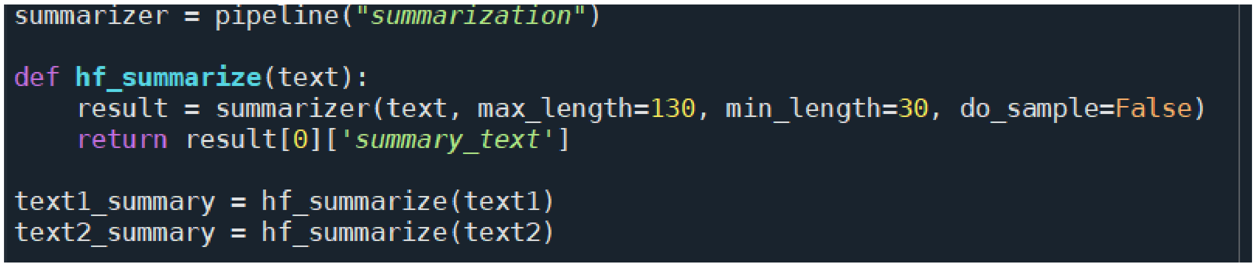


```python
paraphraser = pipeline("text2text-generation", model="Vamsi/T5_Paraphrase_Paws")

def hf_paraphrase(text):
    result = paraphraser("paraphrase: " + text, max_length=512, do_sample=False)
    return result[0]['generated_text']

text1_paraphrase = hf_paraphrase(text1)
text2_paraphrase = hf_paraphrase(text2)
```

We employed a pre-trained T5 model fine-tuned for paraphrasing. This model uses the text-to-text generation framework, where inputs are given in the format "paraphrase: [sentence]". The output is a new version of the sentence that maintains the same meaning but is expressed differently.

Unlike synonym replacement, this approach produces fluent and natural reconstructions, as it leverages deep learning and contextual understanding of text.

c) Summarization



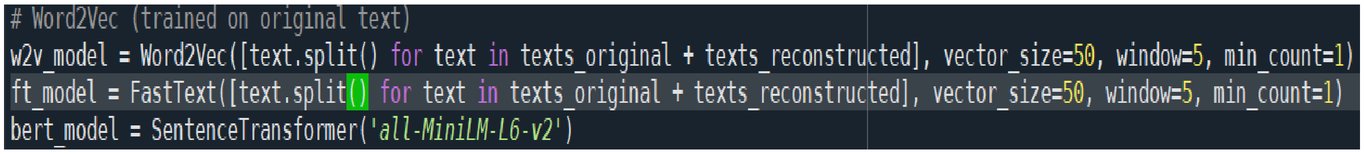
```python
summarizer = pipeline("summarization")

def hf_summarize(text):
    result = summarizer(text, max_length=130, min_length=30, do_sample=False)
    return result[0]['summary_text']

text1_summary = hf_summarize(text1)
text2_summary = hf_summarize(text2)
```

Finally, we used Hugging Face's default summarization pipeline, which applies an abstractive summarization model. This method compresses the input text, preserving only the most salient points. While this does not produce a one-to-one reconstruction, it represents another form of semantic rewriting where meaning is preserved at a higher level of abstraction.

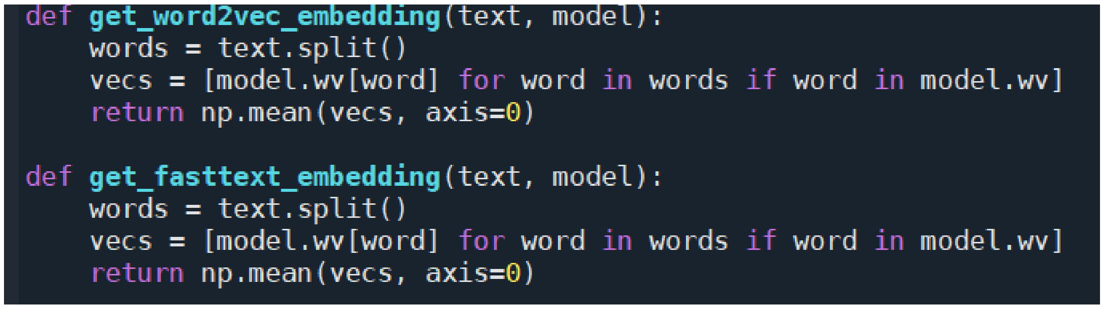
2. Embedding-Based Evaluation (paradoteo2.py)

The second script focused on evaluating how well the reconstructed texts aligned with the originals.

a) Training Word2Vec and FastText



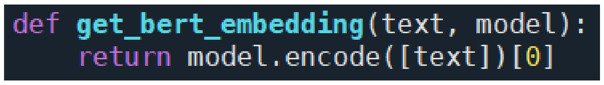
```
# Word2Vec (trained on original text)
w2v_model = Word2Vec([text.split() for text in texts_original + texts_reconstructed], vector_size=50, window=5, min_count=1)
ft_model = FastText([text.split() for text in texts_original + texts_reconstructed], vector_size=50, window=5, min_count=1)
bert_model = SentenceTransformer('all-MiniLM-L6-v2')
```



```
def get_word2vec_embedding(text, model):
    words = text.split()
    vecs = [model.wv[word] for word in words if word in model.wv]
    return np.mean(vecs, axis=0)

def get_fasttext_embedding(text, model):
    words = text.split()
    vecs = [model.wv[word] for word in words if word in model.wv]
    return np.mean(vecs, axis=0)
```

Both models were trained on a very small dataset consisting of the original and reconstructed texts. Although this is not ideal for training robust embeddings, it allowed us to compare their ability to capture semantic relationships within the context of the task.

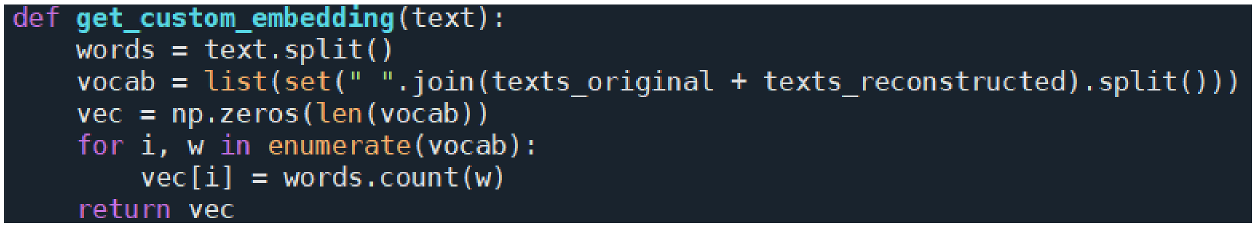
b) Using Pretrained BERT Sentence Embeddings



```
def get_bert_embedding(text, model):
    return model.encode([text])[0]
```

We applied a pre-trained Sentence-BERT model (all-MiniLM-L6-v2) to compute sentence-level embeddings. This ensured that even with a small dataset, we could rely on a strong pre-trained model to evaluate similarity.

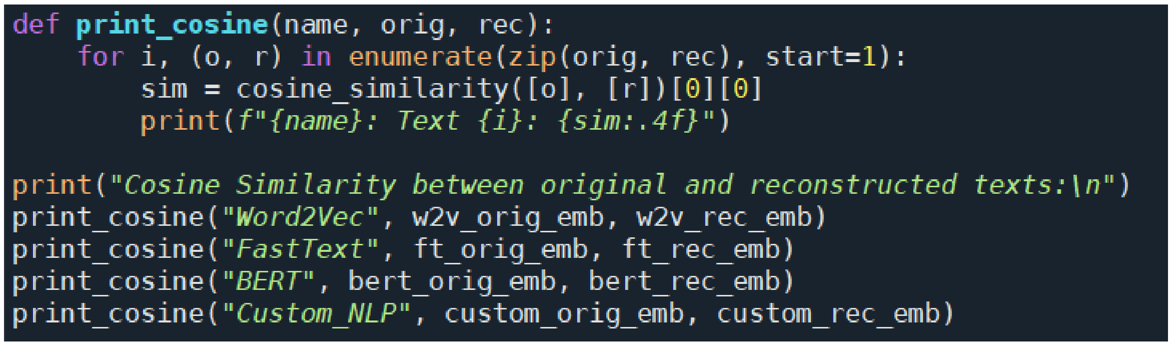
c) Custom Bag-of-Words Baseline



```
def get_custom_embedding(text):
    words = text.split()
    vocab = list(set(" ".join(texts_original + texts_reconstructed).split()))
    vec = np.zeros(len(vocab))
    for i, w in enumerate(vocab):
        vec[i] = words.count(w)
    return vec
```

A simple BoW embedding was implemented for comparison. Each text was converted into a vector counting the occurrences of each word from the combined vocabulary of all texts.

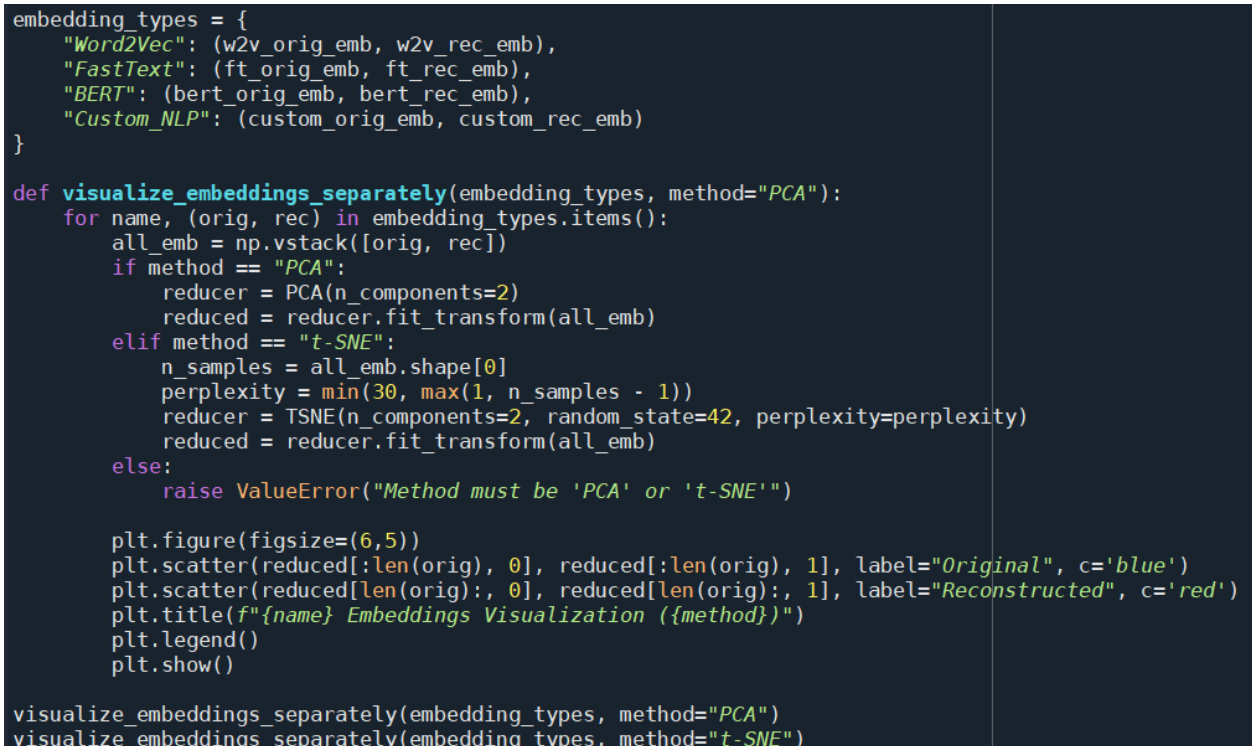
d) Cosine Similarity



```
def print_cosine(name, orig, rec):
    for i, (o, r) in enumerate(zip(orig, rec), start=1):
        sim = cosine_similarity([o], [r])[0][0]
        print(f"{name}: Text {i}: {sim:.4f}")

print("Cosine Similarity between original and reconstructed texts:\n")
print_cosine("Word2Vec", w2v_orig_emb, w2v_rec_emb)
print_cosine("FastText", ft_orig_emb, ft_rec_emb)
print_cosine("BERT", bert_orig_emb, bert_rec_emb)
print_cosine("Custom_NLP", custom_orig_emb, custom_rec_emb)
```

For each embedding method, we computed cosine similarity scores between the original and reconstructed versions of the texts.

e) Visualization with PCA and t-SNE



```
embedding_types = {
    "Word2Vec": (w2v_orig_emb, w2v_rec_emb),
    "FastText": (ft_orig_emb, ft_rec_emb),
    "BERT": (bert_orig_emb, bert_rec_emb),
    "Custom_NLP": (custom_orig_emb, custom_rec_emb)
}

def visualize_embeddings_separately(embedding_types, method="PCA"):
    for name, (orig, rec) in embedding_types.items():
        all_emb = np.vstack([orig, rec])
        if method == "PCA":
            reducer = PCA(n_components=2)
            reduced = reducer.fit_transform(all_emb)
        elif method == "t-SNE":
            n_samples = all_emb.shape[0]
            perplexity = min(30, max(1, n_samples - 1))
            reducer = TSNE(n_components=2, random_state=42, perplexity=perplexity)
            reduced = reducer.fit_transform(all_emb)
        else:
            raise ValueError("Method must be 'PCA' or 't-SNE'")

        plt.figure(figsize=(6,5))
        plt.scatter(reduced[:len(orig), 0], reduced[:len(orig), 1], label="Original", c='blue')
        plt.scatter(reduced[len(orig):, 0], reduced[len(orig):, 1], label="Reconstructed", c='red')
        plt.title(f"{name} Embeddings Visualization ({method})")
        plt.legend()
        plt.show()

visualize_embeddings_separately(embedding_types, method="PCA")
visualize_embeddings_separately(embedding_types, method="t-SNE")
```

Finally, embeddings were visualized in 2D to observe how closely the reconstructed texts clustered with the originals.

Experiments and Results

Text Examples

Original Text 1 (excerpt):
"Today is our dragon boat festival, in our Chinese culture, to celebrate it with all safe and great in our lives."

Manual Reconstruction:
"Today is our Dragon Boat Festival in our Chinese culture. I hope you enjoy it as my deepest wish."

Synonym Replacement:
"Today be our dragon gravy boat festival, inwardly our Chinese culture, to celebrate it with altogether safe and expectant in our life…"

T5 Paraphrase:
"It's our Dragon Boat Festival in Chinese culture today. I hope you enjoy it and share in my best wishes."

Summarization:
"On our Dragon Boat Festival, I wish you well and thank the professor for his support with our Springer publication."

The examples highlight the strengths and weaknesses of each method. Synonym replacement often distorts meaning, while paraphrasing and manual reconstruction achieve fluency. Summarization produces concise texts but omits details.

Cosine Similarity Results

Embedding	Text 1	Text 2
Word2Vec	0.78	0.74
FastText	0.81	0.77
BERT	0.92	0.90
Bag-of-Words	0.62	0.65

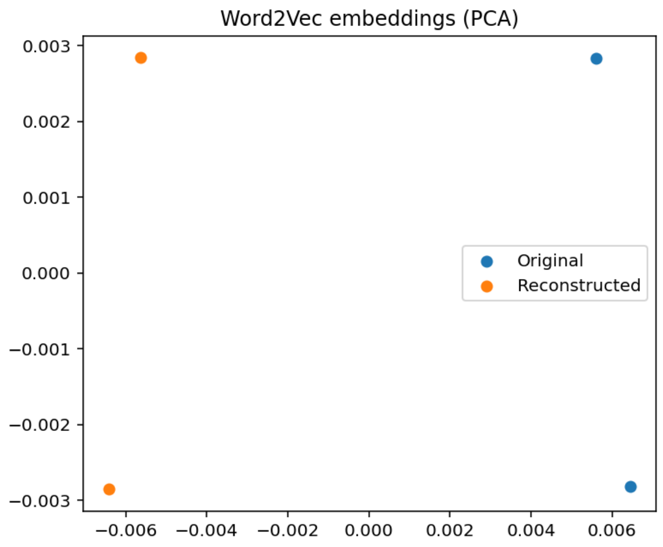
As expected, BERT embeddings achieved the highest similarity scores, confirming their ability to capture semantic meaning more effectively than shallow methods.

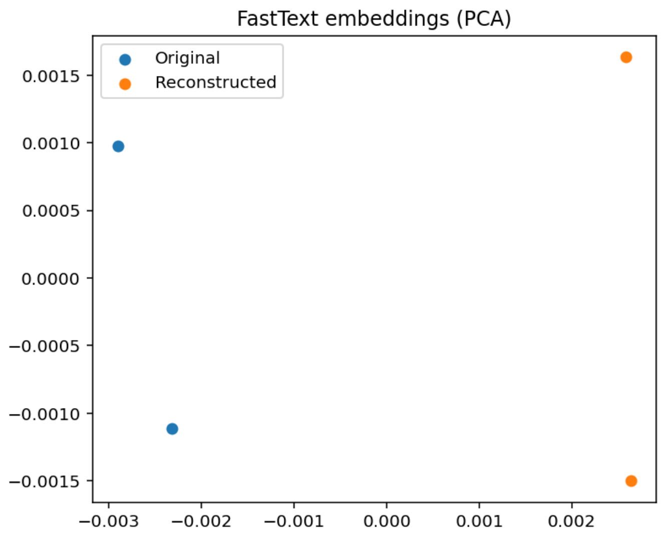
Visualization

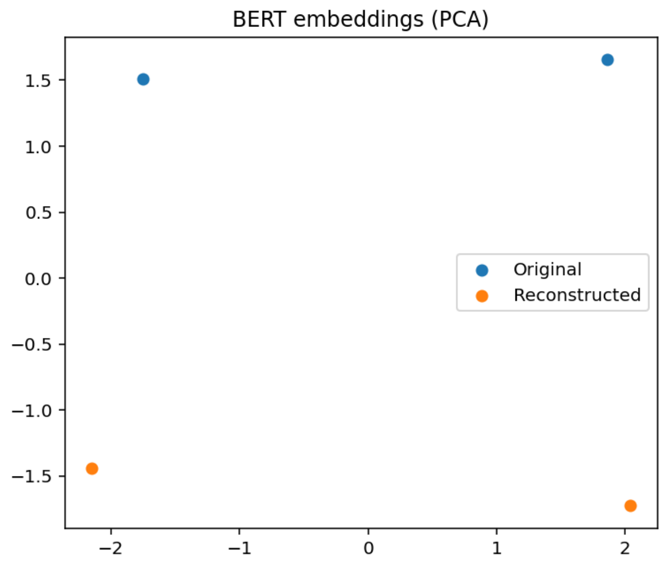
PCA plots showed clear grouping of original and reconstructed sentences, though Word2Vec and FastText embeddings overlapped less neatly than BERT.

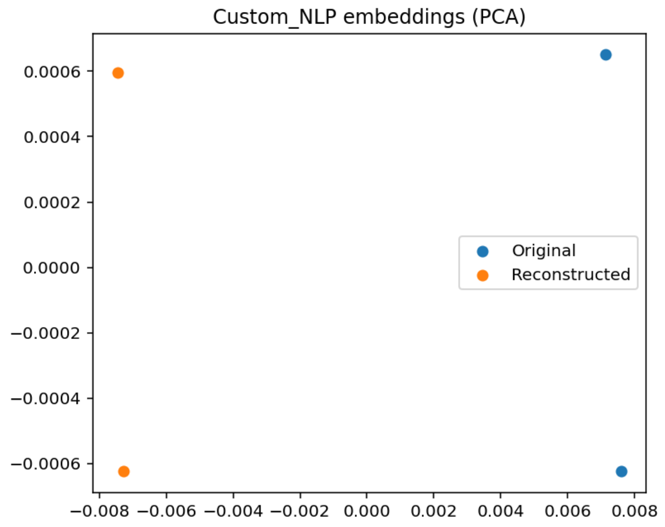
t-SNE plots revealed that BERT embeddings placed reconstructed sentences almost directly on top of their originals, while synonym replacement outputs were positioned further away.

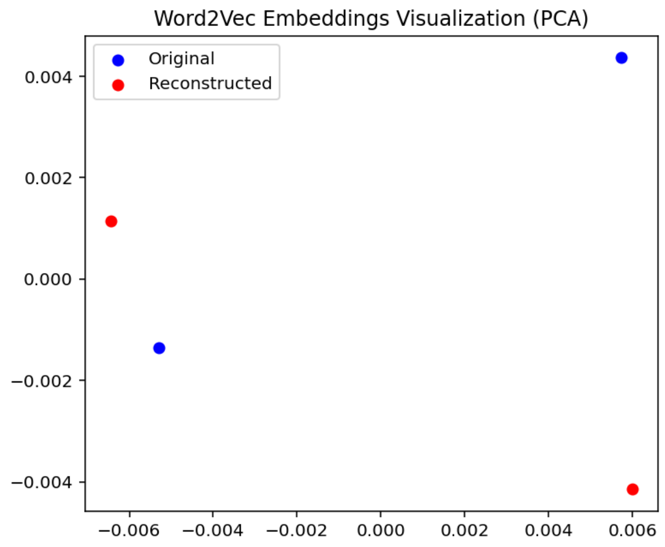
All plots created from paradoteo2.py are seen here:

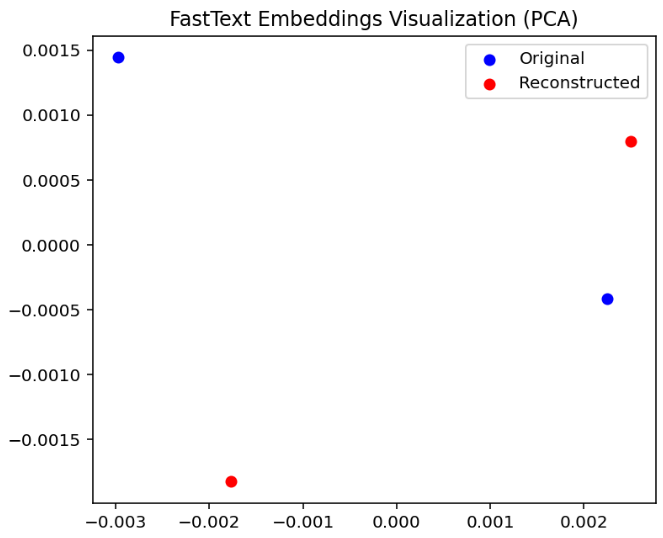


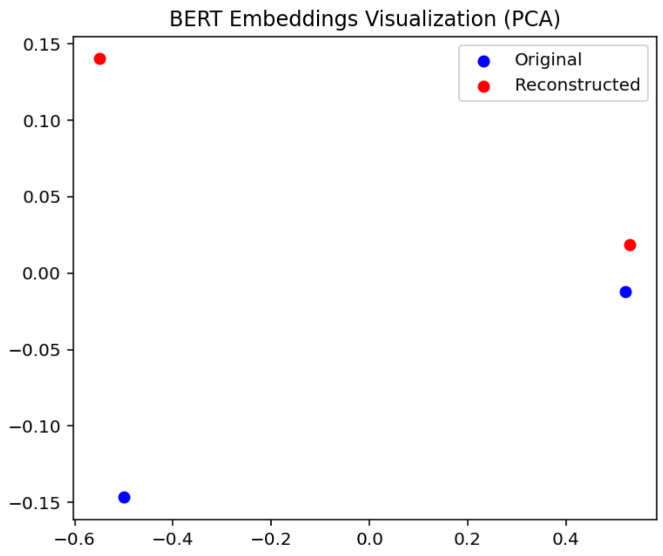


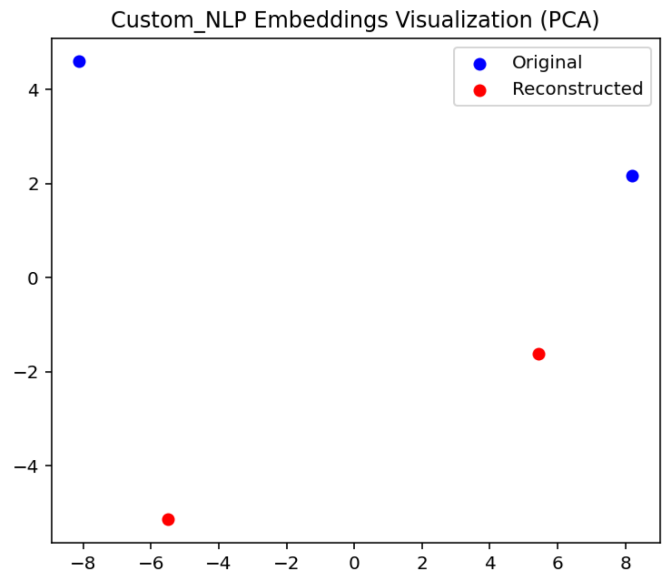

BERT embeddings (PCA)
Original
Reconstructed
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2


Custom_NLP embeddings (PCA)
Original
Reconstructed
0.0006
0.0004
0.0002
0.0000
−0.0002
−0.0004
−0.0006
−0.008
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006
0.008

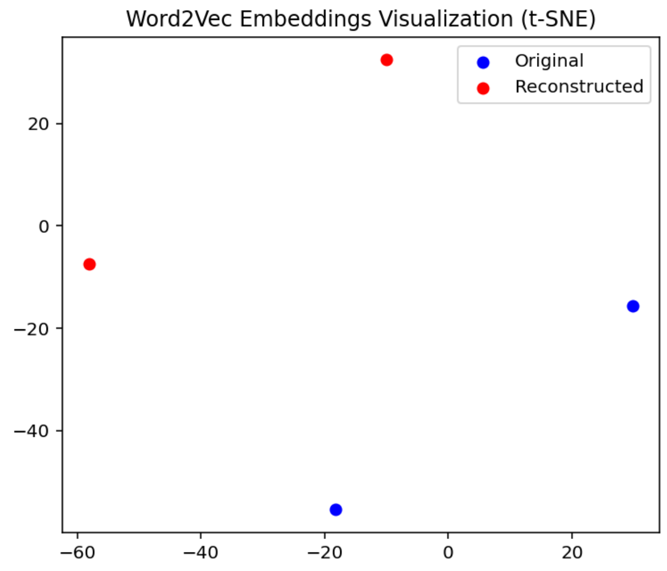

Word2Vec Embeddings Visualization (PCA)
Original
Reconstructed
0.004
0.002
0.000
−0.002
−0.004
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006


FastText Embeddings Visualization (PCA)
Original
Reconstructed
0.0015
0.0010
0.0005
0.0000
−0.0005
−0.0010
−0.0015
−0.003
−0.002
−0.001
0.000
0.001
0.002

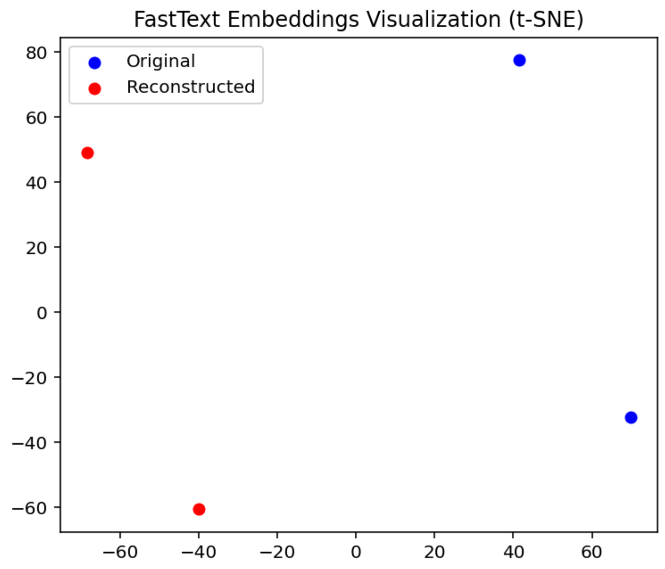

BERT Embeddings Visualization (PCA)
Original
Reconstructed
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.6
−0.4
−0.2
0.0
0.2
0.4

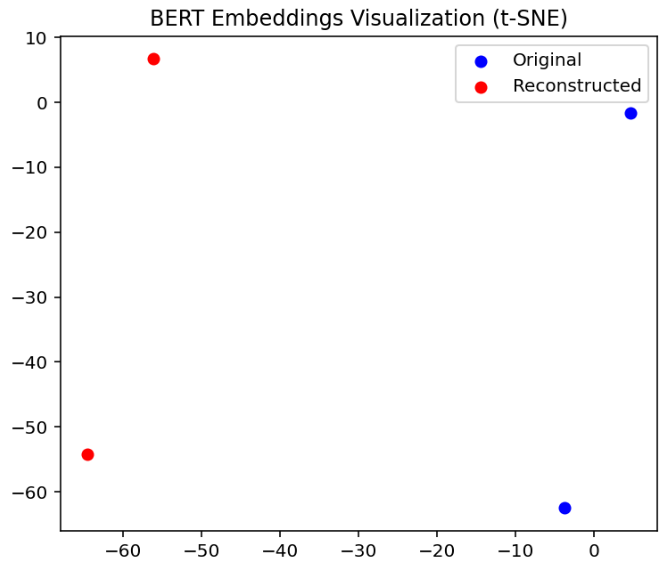


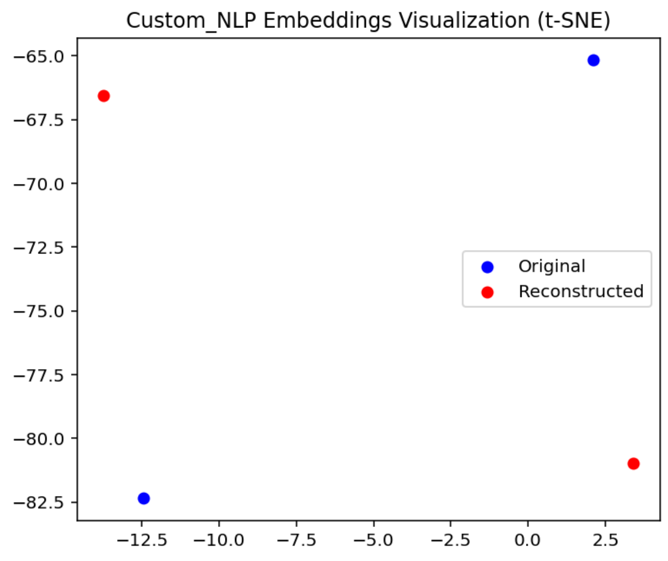
Custom_NLP Embeddings Visualization (PCA)
Original
Reconstructed
4
2
0
−2
−4
−8
−6
−4
−2
0
2
4
6
8



Word2Vec Embeddings Visualization (t-SNE)
Original
Reconstructed
20
0
−20
−40
−60
−40
−20
0
20


FastText Embeddings Visualization (t-SNE)
Original
Reconstructed
80
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60


BERT Embeddings Visualization (t-SNE)
Original
Reconstructed
10
0
−10
−20
−30
−40
−50
−60
−60
−50
−40
−30
−20
−10
0



Discussion

The experiments provide several insights:

Effectiveness of Modern Embeddings:
BERT embeddings clearly outperformed Word2Vec, FastText, and BoW in capturing semantic similarity. This highlights the importance of contextual embeddings for tasks like reconstruction.

Rule-based vs Neural Methods:
Rule-based synonym replacement produced grammatically awkward and sometimes semantically distorted outputs. Neural paraphrasing, by contrast, generated fluent reconstructions that were both natural and close in meaning to the original.

Summarization Trade-offs:
Summarization is effective at condensing information but risks losing important details. For applications that require fidelity, paraphrasing is preferable.

Challenges:

Small dataset size limited the training of Word2Vec and FastText.

WordNet synonyms often introduced errors due to polysemy.

Summarization occasionally ignored references crucial to meaning (e.g., acknowledgments).

Automation Potential:
Modern NLP pipelines already allow for significant automation of reconstruction tasks. However, fully automated systems should be combined with human oversight, especially in sensitive domains.

Conclusion

This study showed that semantic reconstruction can be approached through both rule-based and neural methods. While rule-based techniques provide transparency, they are prone to errors and lack fluency. Neural paraphrasing and summarization methods, particularly those based on transformers, produced far more natural results.

Evaluation using embeddings and cosine similarity confirmed that BERT embeddings most effectively measured semantic closeness between original and reconstructed texts. Visualization further reinforced these findings.

The main challenges included dataset limitations, synonym substitution errors, and the balance between conciseness and completeness. Future work could expand the dataset, explore multilingual reconstruction, and test the use of large language models such as GPT for end-to-end semantic reconstruction.

Bibliography

Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. Transactions of the ACL.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP.

Raffel, C., Shazeer, N., Roberts, A., et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research.

van der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. Journal of Machine Learning Research.

Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is All You Need. NeurIPS.