Reconstruction and Similarity Analysis using NLP pipelines

Team members:
Μαρία Αγγελετοπούλου Π22001
Μύαρης Στυλιανός Π22114
Ναπάτ Ροτζανανόντ Π22150

GitHub: YelAq/NLP_Ergasia_24-25

1.Introduction

In the field of Natural Language Processing (NLP), semantic reconstruction is a crucial task designed to transform unclear, grammatically incorrect, or poorly structured text into language that is clear, coherent, and semantically accurate. This process is particularly important when communicating in multiple languages, writing in non-native English, or in academic settings where language constraints or structural problems might change the original meaning.

This project's goal is to find and analyze different methods for automatically reconstructing two provided texts that have grammatical errors, unclear phrasing, and semantic ambiguity. We try to enhance the original messages' quality and clarity while preserving their intended meaning by utilizing both pre-trained and custom-designed NLP pipelines.

We also evaluate each method's efficiency by comparing the original and transformed texts' semantic similarity. We achieve this using cosine similarity measurements and word embedding models like Word2Vec, GloVe, and BERT. Lastly, we use dimensionality reduction methods like PCA and t-SNE to visualize the changes in semantic space.

2.Methodology

2.1 Reconstruction of 2 sentences using a custom nlp pipeline

From each text sample, we chose to reconstruct the following sentences:

Text 1:

Before: "I am very appreciated the full support of the professor, for our Springer proceedings publication"

After: "I am very grateful for the full support of the professor, for our Springer proceedings publication"

Text 2:

Before: "Anyway, I believe the team, although bit delay and less communication at recent days, they really tried best for paper and cooperation."

After: "Anyway, I believe the team, although a bit delayed and less communicative in recent days, really tried their best to cooperate on the paper."

The approach that we have selected involves the following:

- Breaking down the sentences into individual tokens
- Assigning syntax roles (POS tags) to each of these tokens
- Defining the test (wrong) and the reference (write) versions of each sentence
- Defining and applying specific rules tailored to each sentence

To carry out the reconstruction, the following libraries were also used:

```
import re

import nltk
```

Breaking down sentences into tokens

To acquire the tokens of the test sentence, the regular expression shown in the code below was used. We then passed this pattern and the sentence to be reconstructed into the **re.findall()** function, to extract the tokens into an array called **tokens_test.**

```
#simple regular expression, extract tokens from sentence.

pattern = r"\w+|[.,!?;:'\"-]"

text_test = "I am very appreciated  the full support of the professor, for our
Springer proceedings publication"

tokens_test = re.findall(pattern, text_test)
```

The regular expression pattern “r"\w+|[.,!?;:'\"-]":

This pattern is used to:

· process raw strings (r) (eg. treat backslashes (\) as characters)

· capture one or more words (w+)…

· …Or (|)…

· …capture one of the punctuation marks or symbols that are within the bracket ([.,!?;:'\"-]")

tokens_test has now taken the form:

```
['I', 'am', 'very', 'appreciated', 'the', 'full', 'support', 'of', 'the',
'professor', ',', 'for', 'our', 'Springer', 'proceedings', 'publication']
```

Assigning syntax roles (Part Of Speech tags)

To each token in "***tokens_test***", we assigned a Part Of Speech Tag (POS) tag.

```
#assigning roles to each token.

roles_test = {

    tokens_test[0]: "PRP",      # "I" = Personal Pronoun

    tokens_test[1]: "BVP",      # "am" = Verb, Present, (Non-3rd person
singular)

    tokens_test[2]: "RB",       # "very" = Adverb

    tokens_test[3]: "VBN",      # "appreciated" = Verb, Past Participle

    tokens_test[4]: "DT",       # "the" = Determiner

    tokens_test[5]: "JJ",       # "full" = Adjective

    tokens_test[6]: "NN",       # "support" = Noun

    tokens_test[7]: "IN",       # "of" = Preposition

    tokens_test[8]: "DT",       # "the" = Determiner

    tokens_test[9]: "NN",       # "professor" = Noun

    tokens_test[10]: "PUNC",    # "," = Punctuation

    tokens_test[11]: "IN",      # "for" = Preposition

    tokens_test[12]: "PRP$",    # "our" = Possessive Pronoun

    tokens_test[13]: "NNP",     # "Springer" = Proper Noun

    tokens_test[14]: "NNS",     # "proceedings" = Noun, Plural

    tokens_test[15]: "NN"       # "publication" = Noun

}
```

The same process is carried out with the reference sentence.

```
#The CORRECT sentence

text_reference = "I am very grateful for the full support of the professor,
for our Springer proceedings publication."

tokens_reference = re.findall(pattern, text_reference)
```

After extracting the tokens, (using the same regex pattern), ***tokens_reference*** has now taken the following form:

```
['I', 'am', 'very', 'grateful', 'for', 'the', 'full', 'support', 'of', 'the',
'professor', ',', 'for', 'our', 'Springer', 'proceedings', 'publication', '.']
```

POS tag assignment.

```
#assigning roles to each token.

roles_reference = {

    tokens_reference[0]: "PRP",

    tokens_reference[1]: "BVP",

    tokens_reference[2]: "RB",

    tokens_reference[3]: "JJ",

    tokens_reference[4]: "IN",

    tokens_reference[5]: "DT",

    tokens_reference[6]: "JJ",

    tokens_reference[7]: "NN",

    tokens_reference[8]: "IN",

    tokens_reference[9]: "DT",

    tokens_reference[10]: "NN",
```

```
    tokens_reference[11]: "PUNC",

    tokens_reference[12]: "IN",

    tokens_reference[13]: "PRP$",

    tokens_reference[14]: "NNP",

    tokens_reference[15]: "NNS",

    tokens_reference[16]: "NN",

    tokens_reference[17]: "PUNC"

}
```

The FSA() function

The FSA() function is a Finite State Automata which changes state after each word. The process is as follows:

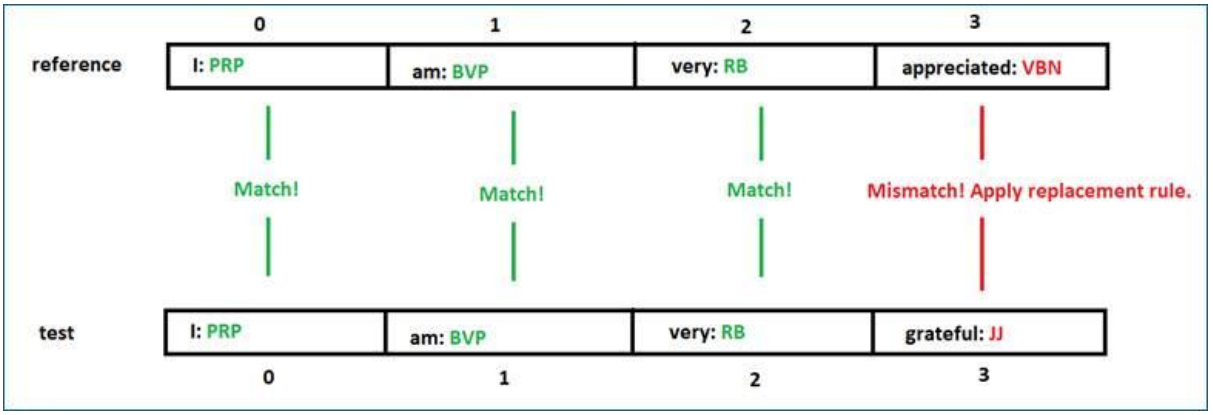
1. **index1 = 0:** This is used to obtain words and POS tags from the roles and tokens arrays. Eg. **roles_reference[tokens_reference[index1]]** is used to get the token at **index1** of **tokens_reference**, and then get the POS of that token through **roles_reference.**

2. **TargetPOS:** This variable contains the POS tags that the word at the current index is supposed to have. It is initialized as **roles_reference[tokens_reference[index1]]**, with **index1** being 0.

3. **Comparing POS tags:** We take the tokens from the reference sentence and the test sentence (at the same index, **index1**) and compare their POS tags. If they are the same, then we transition to the next token of the sentences by incrementing index1 by +1. Else, we apply one of the replacement rules.

4. **Replacement Rules:** There rules are essentially simple if-else statements. For example: If current word is “appreciated”, then replace it with “grateful”. By applying replacement rules whenever there is an error, the **tokens_test** array gradually becomes more and more grammatically correct.

Example image:



Code:

```
index1 = 0

targetPOS = roles_reference[tokens_reference[index1]]    #What POS tag this word is
supposed to have.

def FSA():

    global index1, targetPOS, tokens_test, tokens_reference, roles_test,
roles_reference



    #Append a placeholder at the end, so that tokens_test has a valid
"tokens_test[27]" to work on, so that no errors are caused.

    if index1 == 16 and tokens_test[index1] == "publication":

        tokens_test.append("_")

        roles_test["_"] = "PLACEHOLDER"



    print(f"targetPOS: {targetPOS}")

    print(f"currentPOS: {roles_test[tokens_test[index1]]}")

    print(f"target word: {tokens_reference[index1]}")

    print(f"current word: {tokens_test[index1]}")
```

```
    #Compare targetPOS to the ACTUAL POS that this word has. Are they the same?

    #If so, then there are no errors, so we move on to next word.

    if targetPOS == roles_test[tokens_test[index1]]:

        #Go to next state



        print(f"Length of tokens: {len(tokens_test)}")

        if index1 < len(tokens_test) - 1:

            targetPOS = roles_reference[tokens_reference[index1 + 1]]

            print(f"New targetPOS: {targetPOS}")

        print("Proceeding...")




    #The POS tags are different. Use one of the rules below to fix this error.

    else:

        print("Error found")

        if tokens_test[index1] == "appreciated":  #Rule 1



            print("Rule 1")

            tokens_test[index1] = "grateful"

            roles_test[tokens_test[index1]] = "JJ"

            index1 -= 1     #This is not required. It is only used to show the
corrected version while printing. The fix still takes place regardless.



        elif tokens_test[index1 - 1] == "grateful" and tokens_test[index1] != "for":
#Rule 2



            print("Rule 2")
```

```
            tokens_test.insert(index1, "for")

            index1 -= 1    #This is not required. It is only used to show the
corrected version while printing. The fix still takes place regardless.



        elif tokens_test[index1] == "_" and index1 == 17:   #Rule 3

            print("Rule 3")

            tokens_test[index1] = "."



        else:

            print("No matching rule.")

            print(tokens_test[index1])



    index1 += 1

    print("   ")

    print("   ")
```

The FSA() function is housed in a for-loop, which terminates when we reach the end of the sentence/array.

Next, an **arrayToText(array)** function turns the array back into a string.

```
def arrayToText(array):

    sentence = ""

    finalarray = []

    i = 0

    while i < len(array) - 1:

        finalarray.append(array[i])
```

```
        if POS(array[i + 1]) != "PUNC": #Add a space after a word, ONLY if the
next word is NOT a punctuation. (We want "Hello, World!", and not "Hello ,
World !".)

            finalarray.append(" ")

        i += 1



    finalarray.append(array[len(array) - 1])

    sentence = "".join(finalarray)



    return sentence
```

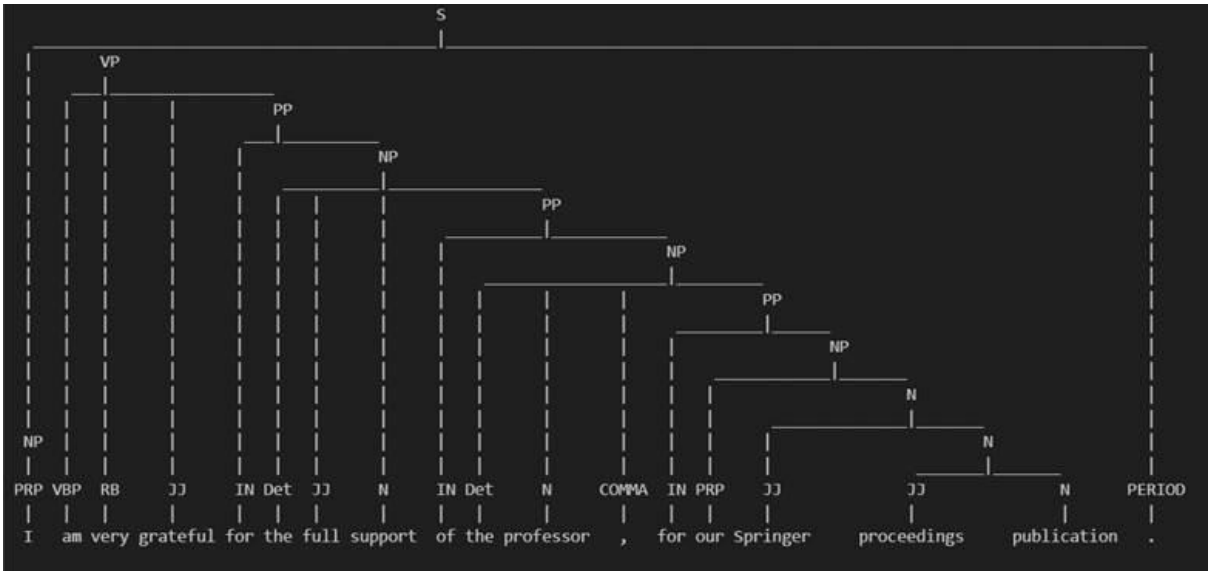
Finally we then define a grammar, so that we can print the syntax tree for this newly reconstructed sentence.

Grammar:

```
grammar = nltk.CFG.fromstring(

    """

    S -> NP VP PERIOD

    NP -> PRP | PRP N | Det N COMMA PP | Det JJ N PP

    VP -> VBP RB JJ PP

    PP -> IN NP

    PRP -> 'I' | 'our'

    N -> 'support' | 'professor' | 'publication' | JJ N

    Det -> 'the'

    JJ -> 'grateful' | 'full' | 'Springer' | 'proceedings'

    VBP -> 'am'

    RB -> 'very'
```

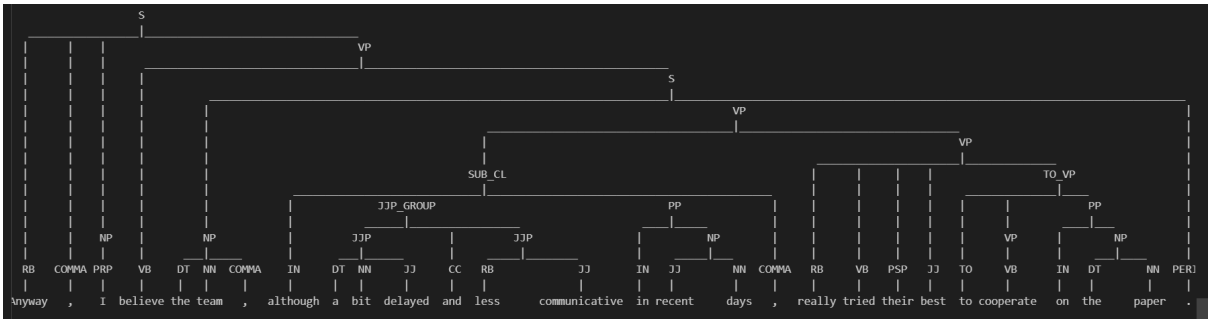
```
    IN -> 'for' | 'of'

    COMMA -> ','

    PERIOD -> '.'



    """

)
```

Resulting Syntax Tree:



This process is carried out for both sentences, the only differences being the replacement rules that were used.

This is the syntax tree for Text 2's sentence.



2.2 Complete Reconstruction of Both Texts with Three Different Automated NLP Pipelines

In Task B, we use three distinct automatic natural language processing pipelines to fully reconstruct the two texts that were provided. The original texts are transformed into three new versions that are clearer, better organized, and semantically accurate by each pipeline using different methods and pretrained models.

Pipeline 1

Using state-of-the-art (NLP) models, we developed a two-step automatic text reconstruction process for the first pipeline to improve semantic clarity and grammatical accuracy.

First, spaCy is used to divide the input text into sentences. After that, a T5-based grammar correction model (vennify/t5-base-grammar-correction) processes each sentence to correct any grammatical mistakes and awkward wording.

The corrected sentences are then fed into a T5-based paraphrasing model (Vamsi/T5_Paraphrase_Paws), which improves understanding while maintaining meaning by producing new sentences that are identical in meaning but more fluid and natural.

The pipeline workflow can be summarized as:

- sentence splitting
- grammar correction
- paraphrasing

Functions used:

```
def split_to_sentences(text):

    doc = nlp(text)

    return [sent.text.strip() for sent in doc.sents]



def grammar_fix(sentence):

    input_text = f"grammar: {sentence}"

    output = grammar_correction(input_text, max_new_tokens=128,
do_sample=False)
```

```
    return output[0]["generated_text"]



def paraphrase(sentence):

    input_text = f"paraphrase: {sentence} </s>"

    output = paraphraser(input_text, max_new_tokens=128,
do_sample=True, num_return_sequences=1)

    return output[0]["generated_text"]
```

Sentence from the first text:

Original : I am very appreciated the full support of the professor, for our Springer proceedings publication

Grammar fixed: I am very appreciated the full support of the professor, for our Springer proceedings publication.

Paraphrased : I am very grateful for the full support of the professor for the publication of our Springer proceeding .

Sentence from the second text:

Original : Also, kindly remind me please, if the doctor still plan for the acknowledgments section edit before he sending again.

Grammar fixed: Also, kindly remind me please, if the doctor still plans for the acknowledgments section edit before he sends again.

Paraphrased : Please also remind me if the doctor still plans to edit the acknowledgments section before he sends again .

Pipeline 2

In this pipeline we used the same workflow as before:

- sentence splitting
- grammar correction

- paraphrasing

Each sentence was first applied to the "prithivida/grammar_error_correcter_v1" model by the pipeline. This model, which is based on a seq2seq transformer architecture, is intended to fix common English grammar mistakes. It was especially successful at:

- adding missing prepositions or articles,
- correcting subject–verb agreement issues

Example:

Input: Anyway, I believe the team, although bit delay and less communication at recent days, they really tried best for paper and cooperation.

After grammar correcting model: Anyway, I believe the team, although a bit delay and less communication in recent days, they really tried their best for paper and cooperation.

The output that had been grammatically corrected was then run through a version of the Pegasus model called "tuner007/pegasus_paraphrase". Its purpose was to produce a sentence with a more natural and logical phrasing that was semantically equivalent.

Example:

Paraphrased: I think the team tried their best for paper and cooperation despite the recent delay and less communication.

When compared to the original input, the pipeline's output showed a noticeable improvement in sentence flow and grammar. It effectively reduced grammatical ambiguity and clarified syntax.However,in some cases, paraphrasing resulted in slightly altered meanings or overly shortened expressions. We will analyze more of that later in the Experiments and Results section.

Pipeline 3

In the third pipeline, we followed the same three-step structure as in Pipelines 1 and 2:

- sentence splitting

- grammar correction

- paraphrasing

However, for this implementation we opted for a more advanced and context-aware language model: GPT-4o-mini, accessed through the OpenAI API. This model is fine-tuned for instruction-following and natural language understanding, which allows it to simultaneously handle grammar correction and semantic paraphrasing with high fluency.

Each sentence is first extracted using the same sentence segmentation function based on spaCy. Then, instead of passing the sentence through separate models for correction and paraphrasing, we submit it as a single prompt to GPT-4o-mini, requesting both grammar refinement and paraphrasing in one step. The result is a rewritten sentence that not only preserves the original meaning but also sounds fluent, professional, and semantically polished.

The pipeline's logic can be summarized as:

- sentence splitting
- instruction-based rewriting using GPT-4o-mini

Functions Used:

```
def split_to_sentences(text):
    doc = nlp(text)
    return [sent.text.strip() for sent in doc.sents]
def gpt4o_rewrite(sentence):
    prompt = (
        f"Fix the grammar, improve clarity, and paraphrase this sentence to sound natural and professional:\n\n{sentence}"
    )
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": prompt}],
        max_tokens=150,
        temperature=0.7
```

```
    )
    return response.choices[0].message.content.strip()
```

Results:

%runfile C:/Users/user/.spyder-py3/NLPassign.py --wdir

Original: Anyway, I believe the team, although bit delay and less communication at recent days, they really tried best for paper and cooperation.

Rewritten: Despite some recent delays and reduced communication, I believe the team has genuinely put forth their best efforts in both the paper and our collaboration.

Conclusions:

Compared to Pipelines 1 and 2, this pipeline produced results that demonstrated not only grammatical accuracy, but also improved fluency, tone, and clarity. In contrast to the seq2seq-based models, GPT-4o-mini showed better awareness of idiomatic language and academic phrasing, while maintaining meaning. Its instruction-following nature allowed us to combine two steps into one, simplifying the pipeline without sacrificing quality.

2.3 Comparison Techniques

Before moving on to cosine similarity and word embeddings, we will compare the results of the four methods (custom + 3 pipelines) based on a syntactic parser we developed.You can find the code on the github repo(parser.ipynb).

We will focus on this original sentence.

Original:”Anyway, I believe the team, although bit delay and less communication at recent days, they really tried best for paper and cooperation.”

Reconstructed Sentences:

Custom Pipeline:Anyway, I believe the team, although a bit delayed and less communicative in recent days, really tried their best to cooperate on the paper.

Pipeline 1:Anyway, I believe that the team, although a bit delayed and less communication in recent days, really tried for paper and cooperation best.

Pipeline 2:I think the team tried their best for paper and cooperation despite the recent delay and less communication.

Pipeline 3:I believe the team, despite some recent delays and limited communication, has made a commendable effort in their work on the paper and in fostering collaboration.

Comparisons based on the results of the syntactic parser(Comparison on full texts for all the pipelines is discussed in section 4.2**):**

Custom Pipeline:The original style and content of the sentence are well preserved. The sentence retains a conversational tone (e.g., “Anyway,” “a bit delayed”) and complex syntax, including the parenthetical and modifying structures present in the original input. Overall, we observe a high-quality reproduction of meaning without significant semantic losses. The sentence feels natural, spontaneous, and human-like.

Pipeline 1:Here we observe that although some improvements have been made (such as introducing “believe that the team”), the irregular and fragmented structure of the original sentence has largely been maintained. Elements like "tried for paper and cooperation best" and the awkward use of "less communication" as a subject reflect a failure to reconstruct the sentence meaningfully. The result is a sentence that appears artificial, grammatically strained, and semantically ambiguous. It reveals that the pipeline may have relied heavily on token-level transformations rather than a deeper semantic or syntactic restructuring. Consequently, the core meaning is distorted and the sentence lacks fluency.

Pipeline 2:The sentence is clearer and shorter in this reconstruction. Despite its simplification, it effectively conveys the main idea: the team gave it their all in spite of delays and communication problems. The pipeline was able to recognize the main relational elements of the sentence and reframe them in a more traditional syntactic structure, as shown by phrases like "despite the recent delay" and "tried their best for paper and cooperation." The original sentence's emotional tone is, however, a little flattened. The sentence sounds more formal because more expressive words like "really tried" and "a bit delayed" are left out. However, it represents a realistic and fairly accurate reorganization.

Pipeline 3:Of all the versions, this one is arguably the most elegantly structured and styled. The sentence is grammatically elegant, lexically rich, and fluid. It employs phrases like "commendable effort," "despite some recent delays," and "fostering collaboration" to present the original message in a cleaned and understandable manner. The underlying sense of effort and cooperation is preserved while a more formal register is introduced through the restructuring. This pipeline provides a semantic paraphrase that improves readability and clarity rather than merely

maintaining the original's words or structure. It displays an extensive understanding of the discourse's underlying intent as well as its surface syntax.

Cosine Similarity and word embeddings(cos_emb_v2.ipynb):

We also developed a code that calculates the cosine similarity of the paraphrased texts compared to the ground truth. The ground truth paragraph used for evaluation was generated with the assistance of ChatGPT.This way we can measure semantic similarity.This code also evaluates the reconstructed texts using BERT embeddings.Using PCA we can visualize how closely the reconstructed tokens align with the ground truth.The results of these techniques are shown in the next section.

3. Experiments and Results

We put the whole 2 texts in all 3 pipelines.(Overall Evaluations of the results is discussed in section 4.2)

Here are the results:

Pipeline 1 (text 1)

```
Today is our dragon boat festival in our Chinese culture to celebrate it with
all safety and great in our lives.Hope you too, enjoy it as my deepest
wishes.Thank you for your message to show our words to the doctor, as his next
contract checking, to all of us.I got this message to see the approved message
.In fact, a couple of days ago I received the message from the professor to
show me this.I am very grateful for the full support of the professor for the
publication of our Springer proceedings .
```

Pipeline 1 (text 2)

```
During our final discussion, I told him about the new submission – the one we
had been waiting for since last autumn – but the updates were confusing as it
did not include the full feedback from the reviewer or maybe the editor?
Anyway, I believe that the team, although a bit delayed and less communication
in recent days, really tried for paper and cooperation best.We should be
grateful, I mean all of us, for the acceptance and efforts until the Springer
link finally came last week, I think .Please also remind me if the doctor
still plans to edit the acknowledgments section before he sends again.Because
I didn’t see that part final yet, or maybe I missed it, I apologize if so.Let
us make sure that all are safe and celebrate the outcome with strong coffee
and future targets .
```

Pipeline 2 (text 1)

Our Chinese culture has a dragon boat festival that is celebrated today.Hope you enjoy it as much as I did.Your message was appreciated by the doctor, as his next contract checks, to all of us.I received the message to see the approved one.I received the message from the professor a couple of days ago.I would like to thank the professor for his full support of the Springer proceedings publication.

Pipeline 2 (text 2)

I told him about the new submission we were waiting for since last autumn, but the updates were confusing as it did not include the full feedback from reviewer or editor.I think the team tried their best for paper and cooperation despite the recent delay and less communication.We should be thankful for the acceptance and efforts until the Springer link came last week, I think.If the doctor still plans for the acknowledgments section to be edited before he sends again, please remind me.I apologize if I missed that part final.Let's make sure all are safe and celebrate the outcome with coffee and targets.

Pipeline 3 (text 1)

Today, we celebrate the Dragon Boat Festival, a significant event in Chinese culture that honors the importance of safety and well-being in our lives.I sincerely hope you enjoy it as much as I do.Thank you for your message. We will present it to the doctor as part of his review of the upcoming contract for our team.I received this message to review the approved content.I received a message from the professor a couple of days ago regarding this matter.I sincerely appreciate the professor's invaluable support for our publication in the Springer proceedings.Let's ensure everyone's safety while we celebrate our achievements with strong coffee and set our sights on future goals.

Pipeline 3 (text 2)

In our final discussion, I informed him about the new submission that we had been awaiting since last autumn. However, the updates were unclear, as they did not include the complete feedback from either the reviewer or the editor.I believe the team, despite some recent delays and limited communication, has made a commendable effort in their work on the paper and in fostering collaboration.We should all express our gratitude for the acceptance and the efforts made until the Springer link was finally received last week.Could you please remind me if the doctor still plans to edit the acknowledgments section

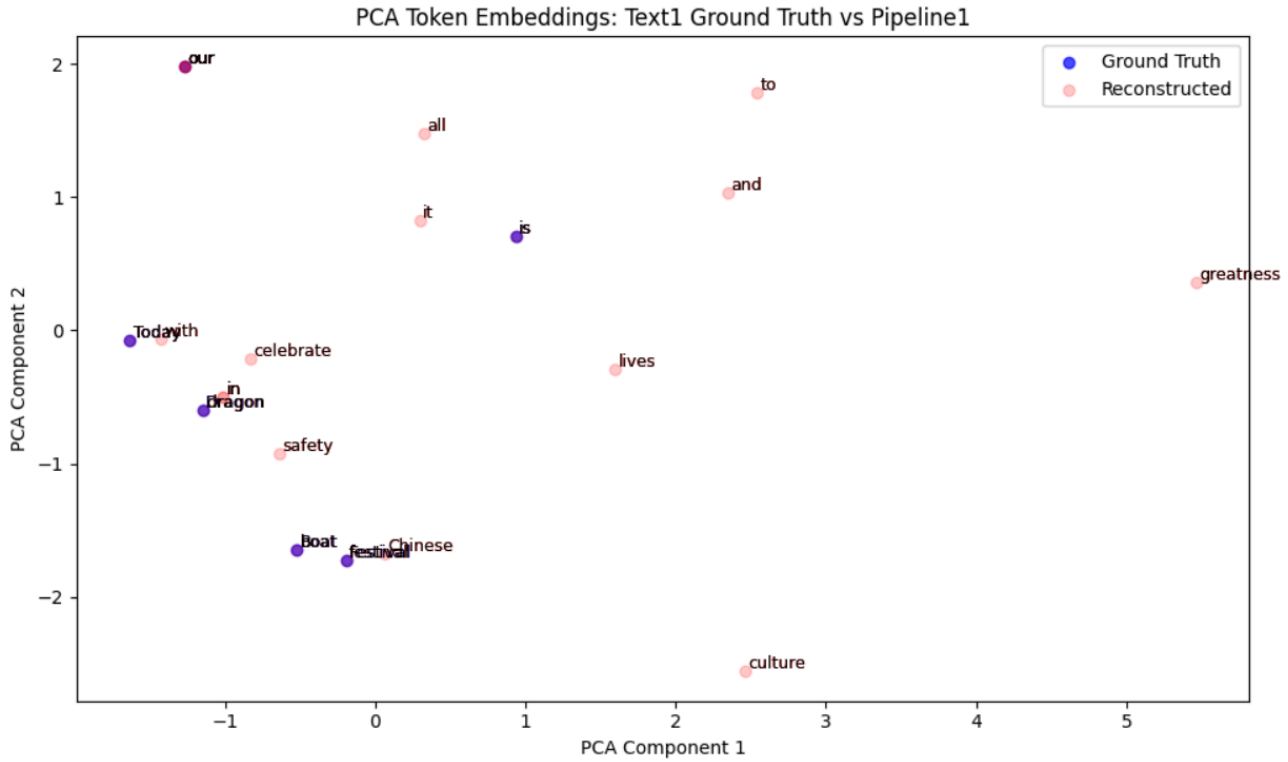
before resending it? Thank you.I apologize if I overlooked that part, as I have not yet seen the final version.

The results of methodology of **Cosine Similarity and word embeddings**

Text	Pipeline	Cosine Similarity
Text1	Pipeline1	0.963393
Text1	Pipeline2	0.956642
Text1	Pipeline3	0.973577
Text2	Pipeline1	0.984293
Text2	Pipeline2	0.970071
Text2	Pipeline3	0.975597

Overall Commentary

With cosine similarity scores ranging from 0.956 to 0.984, all pipelines generated reconstructions that were very similar to the ground truth reconstruction of the original texts. This demonstrates that the reconstruction process was successful overall and that the embeddings were successfully maintained.While Pipeline1 is most effective on Text2, Pipeline3 works best on Text1.



4. Discussion

4.1 Sentence reconstruction with a custom pipeline

What were the biggest challenges during the reconstruction process?

The biggest challenge was deciding which reconstruction approach to choose. There were many reconstruction techniques that required different methods and/or resources. For example, using existing wordbanks, custom word embeddings, machine learning etc. In the end, we decided to simply go with the Rule-based approach.

The next challenge was the definition and implementation of the reconstruction rules. For each grammatical error in each sentence, what would be the correct rule to apply to fix this error? After defining the rule, we then had to implement it in code. Most errors here had to do with arrays and indexes, but they were eventually overcome.

How can this process be automated by using NLP models?

There are many NLP pipelines that already exist that can be used to automate the reconstruction process.For example, some of these were used in 1B to reconstruct

the entire text, and not just one sentence.

4.2 Overall evaluation based on our goal

Pipeline 1

- Text1:Still contains confusing phrasing like “great in our lives”.Feels like a rough rearrangement of the original.
- Text2:Grammar issues(“paper and cooperation best”)

Pros:Close to the original

Cons:Grammar issues,awkward structure,low clarity

Pipeline 2

- Text1:Clearer and more coherent, although it lucks the sentimental tone(warm wishes,appreciation)
- Text2:Better Grammar and flow.This also misses the emotional tone,expressions of gratitude

Pros:Clear,readable

Cons:Oversimplified version of the original,lost meaning in some cases(Your message was appreciated by the doctor)

Pipeline 3

- Text1:The best phrasing out of all 3 pipelines.Keeps the full original message.
- Text2:Professional,natural tone.All key points are kept.

Pros:Clarity,tone,cohesion

Cons:Rephrased more freely.That doesn't actually make it a con but its not really the usual word-for-word rephrasing

---

5. Conclusion

In terms of both language clarity and semantic accuracy, Pipeline 3 repeatedly produced the most precise and organized reconstructions. It naturally improved grammar, coherence, and tone while maintaining the original meaning. Although Pipeline 2 kept the essential elements, it sometimes simplified or changed subtleties. Despite being understandable, Pipeline 1 frequently introduced grammatical errors and semantic shifts that deviated from the original intent. All things considered, Pipeline 3 performed the best at creating versions of the source texts that were comprehensible, coherent, and semantically accurate.

6. References

1.spaCy(en_core_web_sm)

Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.

2.Vennify T5 for Grammar Correction

Model:vennify/t5-base-grammar-correction

Reference:Vennify.(2020). T5 Grammar Correction [Model]. Hugging Face. https://huggingface.co/vennify/t5-base-grammar-correction

3.Vamsi t5 Paraphraser

Model: Vamsi/T5_Paraphrase_Paws

Reference: Vamsi.(2021). T5 Paraphrasing using PAWS dataset [Model].Hugging Face. https://huggingface.co/Vamsi/T5_Paraphrase_Paws

4.Pegasus Paraphraser

Model: tuner007/pegasus_paraphraser

Reference: Zhang,J.,Zhao,Y.,Saleh,M.,& Liu,P.J.(2020). PEGASUS:Pre-training with Extracted Gap-sentences for Abstractive Summarization. In International Conference on Machine Learning(ICML).

https://arxiv.org/abs/1912.08777

5.Prithivida Grammar Correction

Model: prithivida/grammar_error_correcter_v1

Reference:Prithivida. (2021). Grammar Error Corrector [Model]. Hugging Face.

https://huggingface.co/prithivida/grammar_error_correcter_v1

6.Bert for Embeddings

Tokenizer/Model: bert-base-uncased

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

https://arxiv.org/abs/1810.04805

7.Transformers Library

Reference:Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). *Transformers: State-of-the-art natural language processing*. In EMNLP. https://aclanthology.org/2020.emnlp-demos.6

8.Word Embeddings,PCA,Cosine Similarity

General Reference for Word Embeddings and Similarity:Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. arXiv preprint arXiv:1301.3781. https://arxiv.org/abs/1301.3781

9.PCA

Pearson, K. (1901). *On Lines and Planes of Closest Fit to Systems of Points in Space*. Philosophical Magazine.