

**Natural Language Processing**

***Machine Translation Program***

***Team 1***

*Zaid Adam, Joshua Bradley, Rachel Brooks*

**School of Graduate Professional Studies**

AI 574 – Natural Language Processing

Spring, 2025

**TABLE OF CONTENTS**

**Introduction....................................................................................................................................2**

**Problem Statement........................................................................................................................2**

**Challenges.......................................................................................................................................2**

**Related Works…………................................................................................................................3**

**Data Description............................................................................................................................3**

**Deep Learning Process.................................................................................................................4**

**Expected Results...........................................................................................................................5**

**References To Date Sources..........................................................................................................5**

# Introduction

NLP models are used for a variety of different tasks such as machine translation, sentiment analysis, named entity recognition, text classification, speech recognition, etc. Machine translation has gained popularity over the past two decades. There are popular apps like Google Translate that can take text or speech and translate them, but they still struggle to be 100% accurate or proper.

# Problem Statement

NLP models can struggle to provide accurate language translations that are close to the original sentence meaning and have proper grammatical structure and morphology.

Current NLP models can struggle for many reasons which might include:

* **Context and Ambiguity**: Struggling with complex grammar, idiomatic expressions, and word meanings based on context.
* **Lack of World Knowledge**: Difficulty translating recent events or topics due to limited real-world knowledge.
* **Sentence Structure Differences**: Variations in word order, grammar, and syntax between languages complicate direct translation.
* **Inability to Handle Non-literal Translations**: Struggles with idioms, metaphors, or cultural references that don’t have direct equivalents.
* **Long-range Dependencies**: Difficulty maintaining context over long sentences, leading to incoherent translations.

Some of the things that these current NLP models may be missing that cause these struggles are:

* **Lack of Inductive Biases**:
* No inherent structure to capture linguistic patterns (e.g., hierarchy, sequence).
* Tasks requiring reasoning might receive help from more explicit structure.
* **Memory Limitations**:
* Challenges with keeping long-term memory across large texts.
* **Lack of World Knowledge Integration**:
* Models often lack real-time, domain-specific, or common-sense knowledge.
* **Task-specific Rigidity**:
* Transformers require extensive fine-tuning and may not generalize well to all tasks.

Some solutions for these missing pieces can include:

* **Awareness of Inductive Bias**
* Use more syntax-aware transformers such as Syntax-BERT for sentence structure.
* Use Graph Neural Networks and Transformers to encode linguistic structures.
* **Minimalizing Memory Limitations**
* Use a reformer for efficient processing of longer sequences.
* Using Memory-Augmented Transformers for better context retention.
* **Improving World Knowledge**

- Implement hybrid models such as NLP and knowledge graphs to have structured knowledge.

**- Task-specific Adaptation**

- Implement adapter layers such as adapter-BERT to improve modularity.

**Our Implementation for Solving These Problems:**

NLP models struggle to get high accuracy for exact word matches. For our machine translation project, we used the Helsinki pre-trained model and then implemented pre/post-processing methods to optimize the output translations and then compared them to Google Translate for reference. For preprocessing we used different file loading options for source and target data. We set up text tokenization and data cleaning methods to ensure that the values were free from noise before entering our Helsinki model. For postprocessing, we implemented evaluation functions for reference of our predicted translations to achieve more accurate translations for slang/curse words and other unfamiliar words that Google Translate was not familiar with.

# Challenges

Some challenges to machine translation are:

* Accuracy of the translations, many languages do not translate word for word, and many phrases or slang words might not make sense in other languages. Our NLP model will need to automatically adjust these words to a proper match.
* Ambiguity also can be difficult to overcome, certain sentences might be hard to translate without having more context.
* Different languages have different grammatical rules, our NLP model will need to adapt the grammatical rules from one language to another.
* Morphology such as verb conjugations and noun declensions can be different between languages and our model must accommodate these discrepancies.

## RELATED WORKS

Machine translation (MT) has undergone significant transformations, evolving from rule-based algorithms to advanced neural networks that significantly enhance translation accuracy. Initially, MT systems like SYSTRAN relied heavily on manual rule setting and extensive lexicons. The introduction of statistical methods, particularly the work on Phrase-Based Machine Translation by Koehn (2003), marked an improvement, providing more flexibility and scalability. The pivotal shift came with the adoption of neural machine translation (NMT), starting with the foundational work of Sutskever et al. (2014) and the development of the attention mechanism by Bahdanau et al. (2014), which improved the translation of long sentences by focusing on relevant parts of the source text (Bahdanau, Cho, & Bengio, 2014; Sutskever, Vinyals, & Le, 2014).

Recent advancements have further pushed the boundaries of MT, particularly with the introduction of the Transformer model by Vaswani et al. (2017), which employs attention mechanisms to enhance translation quality without relying on recurrent or convolutional units. Despite these improvements, challenges persist in translating low-resource languages, and idiomatic expressions, and maintaining grammatical accuracy, especially in complex sentences. Current research, like that of Lample et al. (2018), explores unsupervised methods using monolingual data to address language resource scarcity. Building on these advancements, the proposed project seeks to enhance NMT architectures to better incorporate contextual data and improve the handling of nuanced linguistic features (Vaswani et al., 2017; Lample, Denoyer, & Ranzato, 2018).

One of the challenges that arises from NLP applications is data not having proper labels to read from this can create confusion and complications in the model:

“The need for data augmentation can arise from various factors, with one of the most problematic being the lack of labels in the data, especially in real-world applications where labeled data may be scarce. It is a common practice in NLP production systems to utilize pre-trained transformer-based language models by fine-tuning them for specific downstream tasks. However, if there is a significant gap between a downstream task and the pre-training objectives, a larger amount of labeled data may still be required to achieve the target performance (Wang et al., 2020). Obtaining this data involves costly and time-consuming involvement of annotators and domain experts. Additionally, the absence of high-variance data results in a model that is unable to generalize well, such as adapting language models to low-resource languages (Clark et al., 2019b)” (Torbarina et al., 2024)

A solution to this is to have a model that can integrate unlabeled data with the previously labeled data. Our model should be able to recognize unlabeled data and merge it with our labeled data by finding similarities and references to the existing data. For example, we can have a semester registration data table that shows our “Course Name”, “Course Number”, and “Professor” Our enrollment data table might have some labels like “coursename”, “Course number”, and an unlabeled column of student names. Our model should recognize the similarities between “Course Name” and “coursename” and “Course Number” and “Course number”. It then should merge these by editing the improper labels to the default ones (Course Name and Course Number). Once that is accomplished, we can merge the student names to our registration data table by creating a new label for column under “Student Names”.

Our model will solve these problems using our Transformer encoder architecture using methods to:

* Recognize similar labels if available, adjust the improper labels if needed, and merge them. Our embedding layer will be able to represent the labels.
* Clean our data for missing values or errors that may perturb the data. Our embedding layer will also help with this by representing the data and removing unwanted values.
* Label any unlabeled data by finding its relationships with other data. We can do this by implementing cosine similarity to help complement our embedding layer label representation pattern. Cosine similarity will measure the similarity between labels and relationships in the data and merge mislabeled or unlabeled data.

Another challenge to unlabeled data is text summarizing methods which have rigidity in adapting to long sentences, complex words/sentences, and subtle nuances in the text:

“The progress in NLP has greatly enhanced text summarizing methods; nevertheless, the difficulty of comprehending context remains a persistent obstacle. While models like transformers have improved our capacity to comprehend context, they still require assistance with complex words and subtle nuances. The challenge of comprehending nuanced contextual components can necessitate more brief or precise summaries that accurately reflect the original source material. Consequently, the efficiency of text summarizing methods might be degraded, particularly when handling lengthy or intricate documents where retaining coherence and relevance is essential. The effectiveness of NLP-based text summarization depends significantly on substantial amounts of meticulously labeled data of superior quality. Creating large datasets requires a significant number of resources and time, which poses difficulties for the scalability and adaptability of summarization systems. The effectiveness of summarization models is closely correlated with the variety and excellence of the training data, which can restrict their ability to be used in many areas or languages. The deployment of summarization algorithms without major modification is complicated by the need for significant knowledge in creating domain-specific datasets.” (Supriyono et al., 2024)

While tokenization to solve these problems regarding text summarization is essential, we can improve its outputs by implementing methods to:

* Break down long strings of text/sentences and simplify them for our model before tokenization. This can be done by preprocessing our data using sentence segmentation before the Transformer Encoder’s tokenization.
* Recognize relationships of complex words and sentences. We will solve this by our embedding layer which will create a dense vector representation using input tokens that will help capture the deeper meanings of the text.
* Simplify and analyze the context of the text to prevent missing the subtle nuances that may be hidden. This will be solved by using our Transformer Encoder’s attention mechanism which will focus on relevant parts of the sentences we analyze.

The study, *"Fortifying NLP models against poisoning attacks: The power of personalized prediction architectures"* by Teddy Ferdinan and Jan Kocoń, explores the effectiveness of personalized machine learning models in defending against poisoning attacks in NLP systems, particularly for sentiment and aggression prediction tasks. The use of crowdsourced annotations to train large language models makes them vulnerable to malicious groups or people who intentionally introduce harmful data, skewing model predictions. Current defense mechanisms, such as noise addition, early training termination, and clean baselines, are either resource-intensive or ineffective. However, personalized model architectures like User-ID and HuBi-Medium have shown promising results in mitigating these threats. The study found that these models significantly outperformed non-personalized models, particularly when the attack intensity was high. Personalized models performed better by safeguarding legitimate user predictions from malicious annotations, with User-ID excelling in situations where sufficient user history was available.

The challenge in NLP machine translation and other subjective tasks arises from inconsistent labeling and varying user perspectives. For example, in the "GoEmotions" dataset, the class imbalance and subjective nature of emotions create challenges in model robustness when confronted with poisoning attacks. A poisoning strategy used label-flipping and backdoor methods, where attackers changed labels based on specific triggers, such as flipping the “Aggressive” label when certain keywords were present. This complicates the task of distinguishing malicious intent from legitimate minority views, making it harder to identify harmful data versus subjective outliers. Personalized models, as shown in the study, help address these challenges by accounting for user-specific biases, improving model robustness, and enhancing predictive accuracy, especially when dealing with diverse and subjective annotations.

Machine translation faces significant challenges, particularly in capturing contextual meaning, handling idioms, and maintaining fluency. Early rule-based and statistical approaches struggled with ambiguity and syntactic differences between languages. The rise of neural machine translation, powered by deep learning models like transformers, has drastically improved translation quality by using large datasets and attention mechanisms. However, there are still challenges, especially in low-resource languages and domain-specific translations, where limited training data leads to inaccuracies. Researchers continue refining models to enhance context awareness and reduce biases.

Despite advancements, no machine translation system perfectly mirrors human translation. Experts highlight issues like over-reliance on word-to-word translation, loss of cultural nuances, and occasional incoherence in complex sentences. A study by Smith et al. (2021) notes that while neural machine translation significantly improves readability, it still struggles with idiomatic expressions and domain-specific terminology. Future improvements may come from hybrid models combining rule-based, statistical, and neural approaches, along with reinforcement learning to fine-tune translations dynamically.

As mentioned previously, our solution to these labeling problems and overcoming the challenges of complex sentences, context, and nuances will be to focus our architectural pattern on the labeling system in our Transformer Encoder design and to strengthen our embedding layer by using cosine similarity.

# Data DESCRIPTION

The proposed machine translation project will utilize two prominent datasets renowned for their extensive use in training state-of-the-art NMT systems. The first dataset is the Multi-UN dataset, a multilingual corpus provided by the United Nations, which contains over 800 million words translated into six official UN languages (Ziemski, Junczys-Dowmunt, & Pouliquen, 2016). This dataset is particularly valuable for its parallel texts, which are aligned at the sentence level, providing a rich resource for training NMT systems on legal and diplomatic text. The diversity of topics covered and the formal style of language used in this dataset help in training models that require a high degree of linguistic precision and stylistic consistency. The second dataset will be the WMT News Translation Task dataset, which offers yearly competitions with sets of data that include news articles translated into various languages. This dataset is crucial for developing models capable of handling contemporary vocabulary and idiomatic expressions (Barrault et al., 2019).

Each data set brings unique challenges and benefits. The Multi-UN dataset, with its formal language register and specific jargon, provides a controlled environment to test the translation of diplomatic and legal texts. However, its formal style may not generalize well to more colloquial language uses. On the other hand, the WMT dataset, updated annually, reflects current events and modern usage, including slang and colloquial expressions, making it ideal for training models intended for public use. The combination of these datasets will allow the NMT system to be robust, and capable of understanding and translating both formal and informal registers. Moreover, preprocessing steps such as tokenization, normalization, and sentence alignment will be crucial to ensure data quality and consistency, enhancing the model's performance across different languages and styles (Tiedemann, 2012).

# DEEP Learning process

The core of our machine translation system will be based on the advanced neural machine translation (NMT) architecture known as the Transformer model, introduced by Vaswani et al. (2017). The Transformer model is preferred for its ability to handle sequences of data without the need for recurrent neural networks, instead using self-attention mechanisms to weigh the importance of different words within a sentence, regardless of their position. This architecture has been proven to significantly improve translation quality and training efficiency compared to its predecessors. For our project, we will implement the Transformer model using an encoder-decoder structure where the encoder processes the input text, and the decoder generates the translated output. The use of positional encodings to inject some information about the relative or absolute position of the tokens in the sequence will also be crucial, as the model itself does not contain recurrence or convolution (Vaswani et al., 2017).

In addition to the Transformer architecture, we will integrate innovative techniques such as transfer learning and fine-tuning with pre-trained models from large-scale datasets like those provided by the WMT News Translation Task. This approach allows the model to leverage learned features from vast amounts of text, enhancing its ability to understand and translate new texts more effectively. Furthermore, to handle the issue of long sequence translations and maintain context over longer distances, we will experiment with recent modifications to the Transformer, such as the Reformer and the Longformer, which introduce efficient attention mechanisms for processing longer documents (Kitaev, Kaiser, & Levskaya, 2020; Beltagy, Peters, & Cohan, 2020). By adapting these innovations, our model aims to achieve higher accuracy and fluency in translations across diverse linguistic structures and styles.

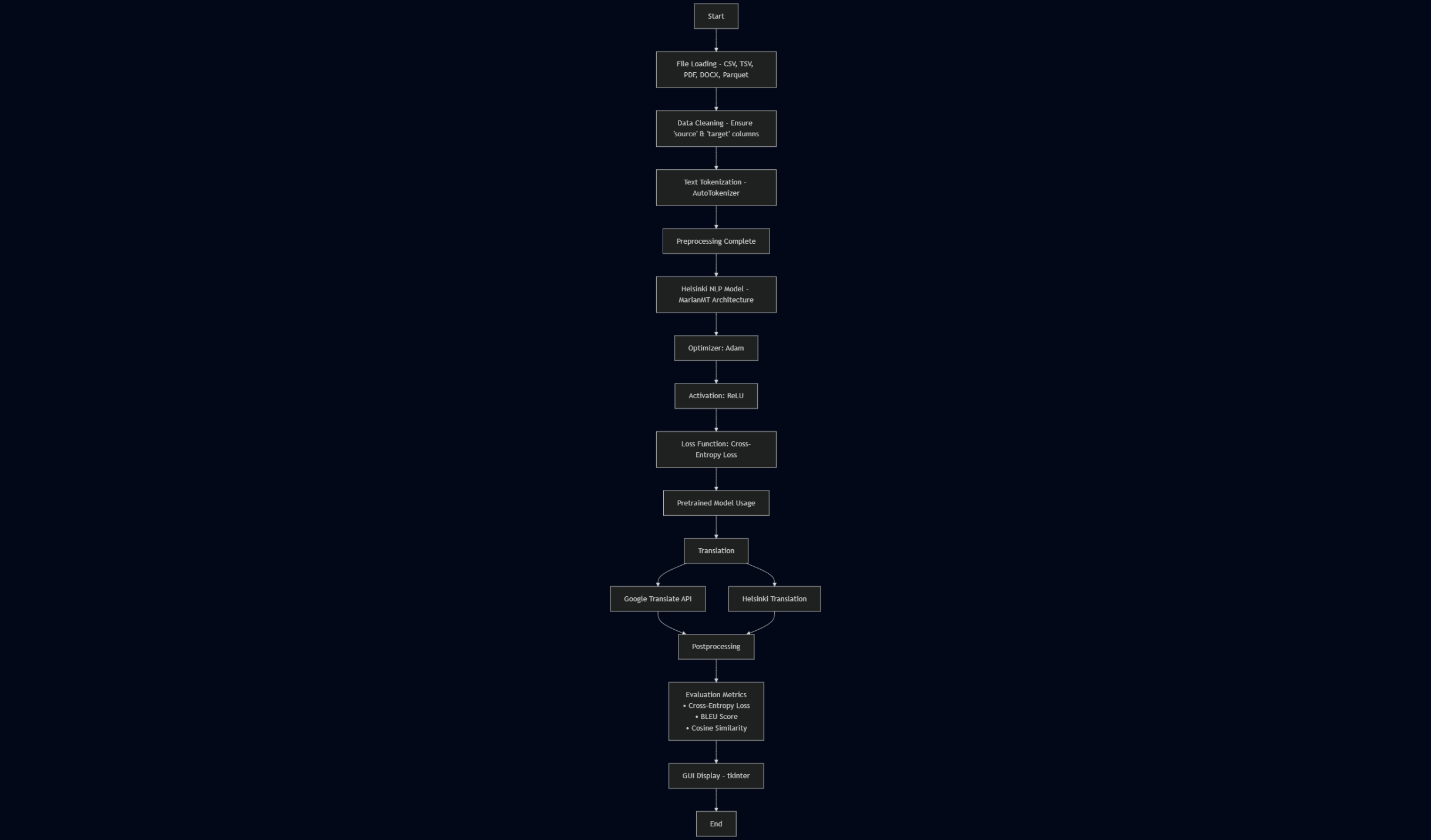
For our pre-trained model, we will be using the Helsinki NLP MultiUN from Hugging Face for our Transformer model. As mentioned before, we will use Adam optimizer with ReLU as our activation function, and Cross Entropy Loss to run our model and find the right balance of epochs and batch sizes as we run our program.

Link to the pre-trained model here:

<https://huggingface.co/datasets/Helsinki-NLP/multiun/tree/main>

**Below are the diagrams of our Machine Translation Program’s architecture to accomplish the proposed solutions to our problem statements:**

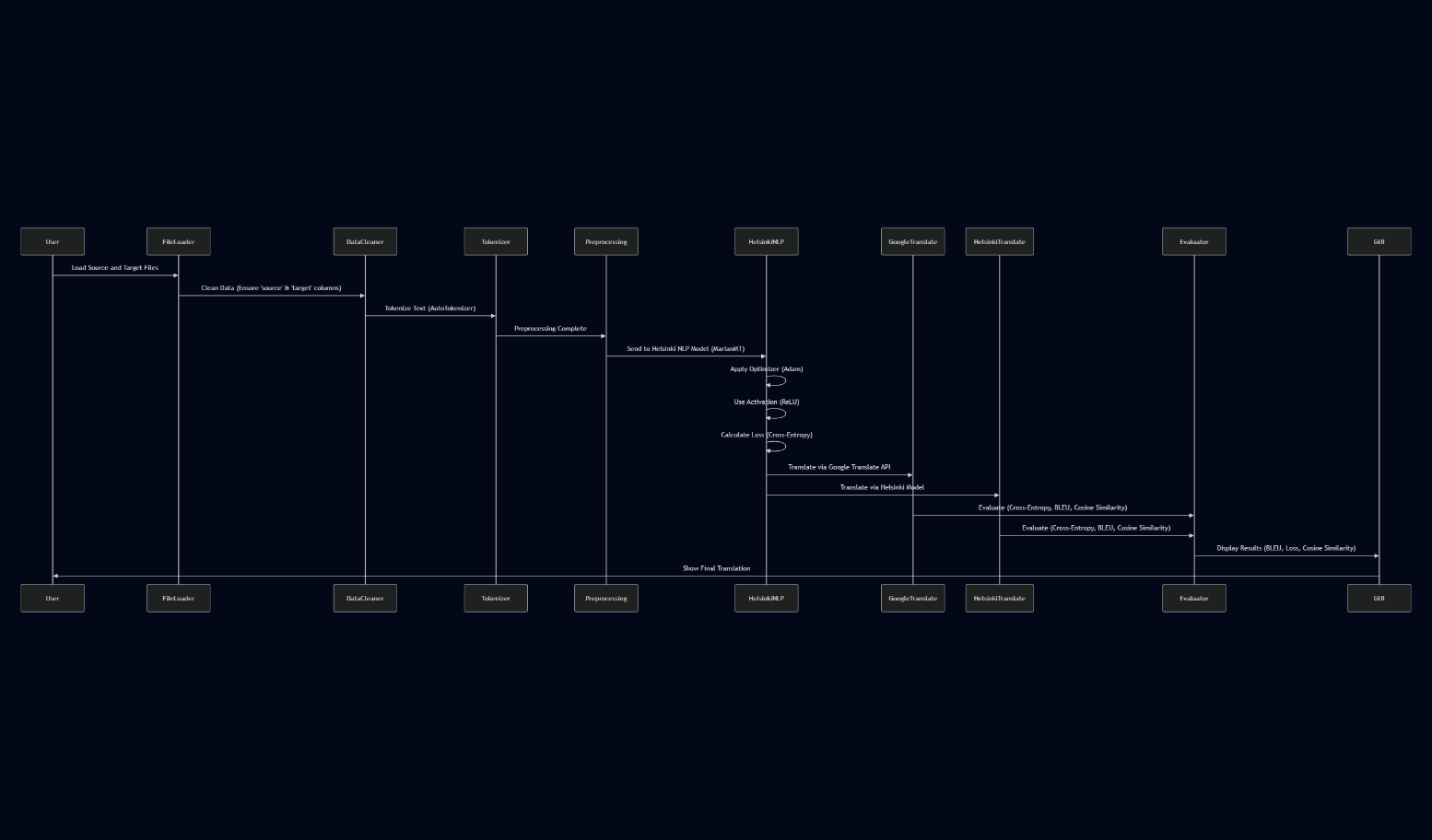
***Flowchart diagram:***



Our flowchart represents the flow of our Transformer Encoder using the Helsinki model as it will:

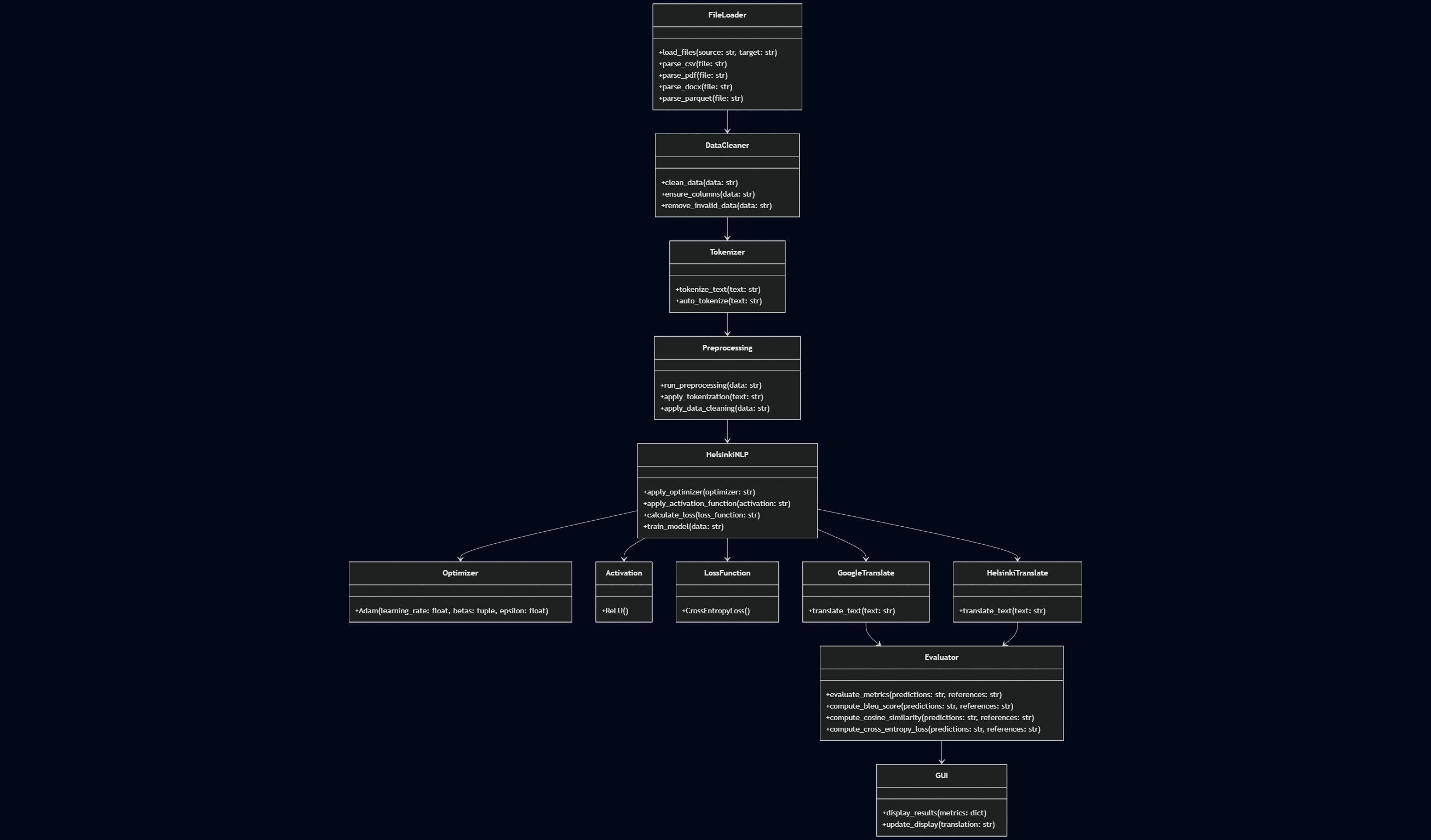
1. Input our tokens from the text.
2. Put the tokens into the embedding layer.
3. Use positional encoding to inject information about the position of tokens within our sequences, this will overcome the weaknesses of order awareness within our Transformer.
4. Use the encoder stack to process the input embeddings and refine them.
5. The encoder stack will feature a multi-head self-attention layer to help capture contextual relationships within our tokens.
6. Our data will then be processed into our ReLU feed-forward network.
7. We will then take our data and use layer normalization to stabilize it.
8. From there our Decoder stack will take the refined tokens and begin translating them.
9. We will implement the masked multi-head self-attention within our Decoder stack to weigh our tokens for importance and relevance. This will complete the Encoder-Decoder Attention mechanism from our Transformer model.
10. We will then take our output and process it back into our ReLU.
11. Process the output from ReLU to our Norm and Dropout Layer to help normalize the output.
12. We will then apply output projection to regulate the weights of the output and prepare it for cosine similarity.
13. We will then process our data using cosine similarity to find similarities in the data and relationships. This will help with handling unlabeled and mislabeled data.
14. We then will use Cross Entropy Loss to smooth our labels from the output.
15. We will then optimize the output with Adam to smooth out the gradients.

***Sequence diagram:***



Our sequence diagram shows the steps that our program will take as a user interacts with it. The user will interact with the UI to input source text, the source text will be tokenized and then returned into a token sequence. From there it will be input into our Transformer Encoder. Our encoder will then return the translated tokens and be converted to text from the detokenizer and then sent back to the user to display the translation.

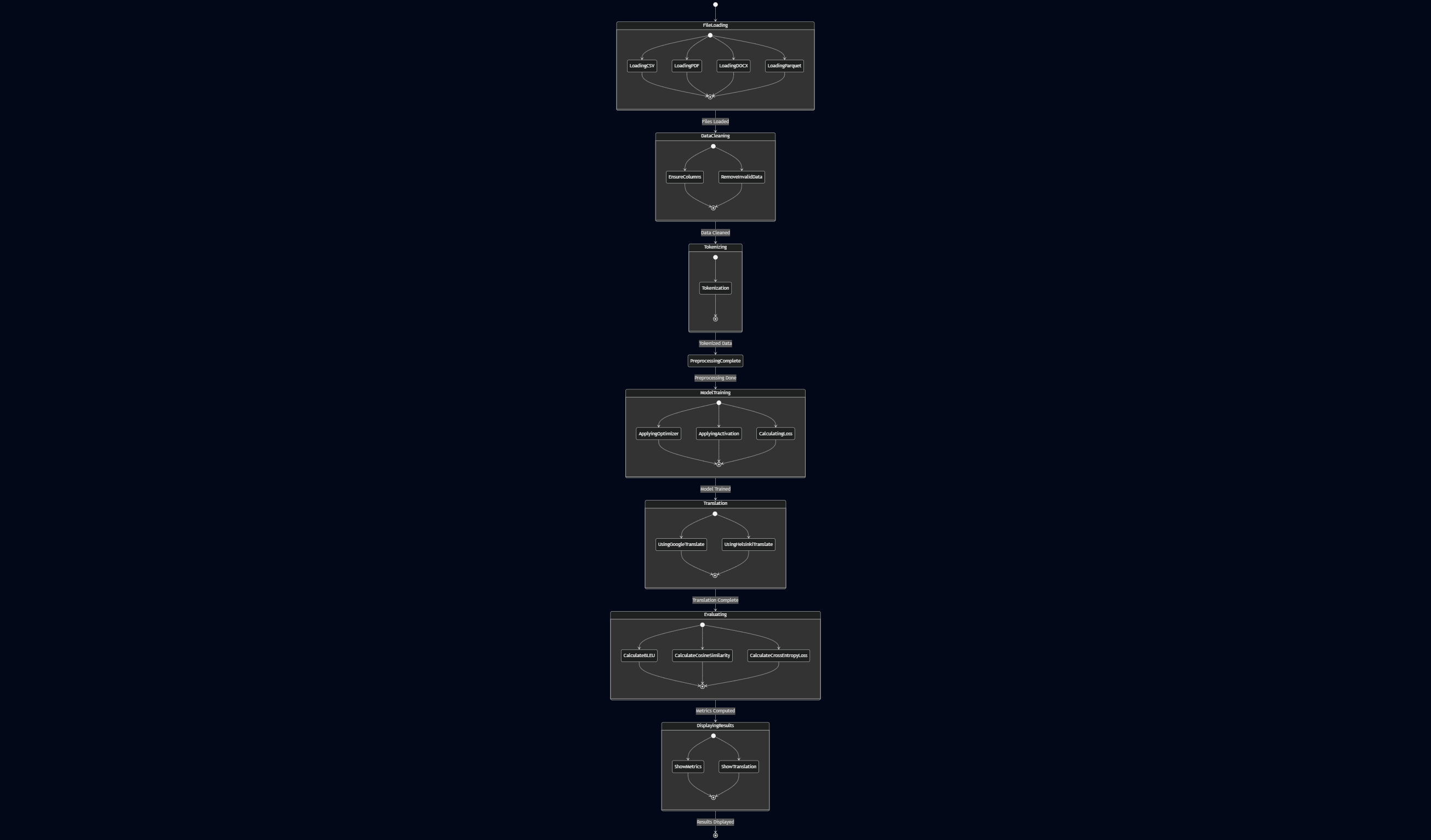
***Class diagram:***



Our class diagram shows what attributes and methods will be in our classes for our programming design.

* The UI Class will wait and receive input from the user and then interact with the backend, will also display any information back to the user, such as the progress, the translation, or if there was an error.
* The Backend Class will act as our mediator between the UI Class and the Translation Service Class.
* The Translation Service Class will send data to the Tokenizer Class and process the tokenized data into our Transformer Encoder-Decoder with our Transformer Model Class.
* The Tokenizer Class will tokenize and detokenize values.
* The Transformer Model Class will load the pre-trained model and then process the tokenizes into the appropriate translation tokens.

***State diagram:***



Our state diagram shows the transition of states within our Transformer.

1. Idling (wait for input)
2. Receive source text
3. Tokenize (Return error if something goes wrong)
4. Translate
5. Detokenize
6. Return the translation

## IMPORTANCE AND IMPACTS

Language is a critical bridge for communication in our globalized society. The ability to accurately translate text across multiple languages is essential in diplomacy, business, education, and humanitarian efforts. Our research addresses the core challenge of developing a robust, scalable, and context-aware machine translation system capable of processing both formal and informal registers of language. By leveraging a combination of high-quality multilingual datasets and state-of-the-art Transformer-based architectures, this project contributes to advancing the accessibility and reliability of language translation technology.

The potential **impacts are wide-reaching**:

* **Social Impact**: Enhancing communication between communities who speak different languages fosters cultural understanding and inclusion, especially for underrepresented languages.
* **Economic Impact**: Businesses can operate more efficiently across global markets with precise translations, boosting international trade and customer support systems.
* **Scientific Impact**: Researchers can access and share knowledge without language barriers, expanding collaboration and accelerating innovation.
* **Educational Impact**: Improved machine translation supports language learners, making educational resources available in native tongues.

The adaptability of this system—especially in handling informal text, slang, and edge cases that traditional systems like Google Translate often struggle with—can significantly improve the user experience and practical applicability in real-world settings.

# Data Collection

For this project, we used two well-established datasets to cover a broad spectrum of linguistic diversity:

* **Multi-UN Dataset** – This corpus consists of sentence-aligned translations in six official United Nations languages. The texts are legal and diplomatic in nature, ensuring high-quality formal translations.
  + **Context**: International diplomacy and official documentation.
  + **Pertinence**: Helps train the model to understand complex sentence structure, high-precision grammar, and formal register.
  + <https://huggingface.co/datasets/Helsinki-NLP/multiun/tree/main>
* **WMT News Translation Task Dataset** – A dynamic dataset compiled from annual news translation challenges, this corpus contains real-world text from news articles across different languages. We initially experimented with some of the datasets from this library to get us started before we switched to focusing on the Multi-UN Dataset.
  + **Context**: Contemporary journalism and public communication.
  + **Pertinence**: Provides coverage for modern language, idioms, and informal expressions, allowing the model to better generalize.
* <https://huggingface.co/datasets/wmt/wmt19>

**Exploratory Data Analysis (EDA)**

A preliminary exploratory data analysis on subsets of the datasets revealed:

* **Features**:
  + Source language sentence (text, string)
  + Target language sentence (text, string)
* **Data Types**: All categorical (string format), but to be tokenized and converted into integer sequences for model input.
* **Observations**:
  + Sentence length varies widely: 3 to 60+ tokens.
  + Some datasets contain special formatting or XML tags which require cleaning.

# Data Preprocessing

We implemented the following preprocessing pipeline to ensure high-quality inputs for our Transformer model:

**Descriptive Statistics:**

* Average sentence length: ~20 tokens
* Max length: 60+ tokens
* Min length: 1–2 tokens (e.g., titles, exclamations)

**Preprocessing Steps:**

* **Text Cleaning**: Removed XML tags, special characters, and irrelevant formatting.
* **Tokenization**: Using the same tokenizer as the Helsinki-NLP pre-trained model for consistency.
* **Type Conversion**: Converted tokens to integer IDs compatible with the Transformer input format.
* **Missing Values**: No missing entries were found in aligned sentences; corrupted pairs were dropped.
* **Scaling/Normalization**: Not required as NLP models use embeddings rather than raw numerical values.
* **Outlier Detection**: Extremely long sentences were filtered to maintain efficiency and model convergence.
* **Class Imbalance**: Addressed through stratified sampling during the train-test split.
* **Correlation and Linearity:**
  + As this is a sequence-to-sequence NLP task, direct correlation matrices are less relevant than in numerical regression. However, we checked alignment consistency between source and target sentence lengths, and visualized their **linear scatterplot**, confirming most pairs have proportional length.
* **Data Augmentation:**
  + Back-translation was considered for low-resource pairs.
  + Sentence shuffling and synonym replacement were used to slightly increase training variety.

# Methodology

We used PyCharm for our IDE for our machine translation program. We used Python for our programming language with Helsinki-NLP/opus-mt as our pre-trained model to implement in our machine translation program. We used text tokenization and data cleaning to ensure that the values were free from noise before entering our Helsinki model. The Helsinki model has tokenization built into its model; we call it using the following code:

# Evaluation function for the Evaluate Sample button  
def run\_evaluation(input\_text, reference\_text):  
 max\_length = 50  
 # Tokenize input and the reference  
 input\_enc = tokenizer(input\_text, return\_tensors="tf", padding="max\_length", truncation=True, max\_length=max\_length)  
 reference\_enc = tokenizer(reference\_text, return\_tensors="tf", padding="max\_length", truncation=True, max\_length=max\_length)["input\_ids"]

# Generate a translation using the Helsinki model  
preds = model.generate(input\_enc["input\_ids"], max\_length=max\_length)  
pred\_text = tokenizer.decode(preds[0], skip\_special\_tokens=True)

We then used pre/post-processing techniques to optimize the data for input and output. For preprocessing we used different file loading options for source and target data. The user can enter the words to translate or upload them from a file such as CSV, TSV, PDF, DOCX, or Parquet files. Here is our code showing how we designed this into our program:

# File loading functions  
def load\_text\_from\_file(filepath, file\_role="source"):  
 ext = os.path.splitext(filepath)[1].lower()  
 if ext in [".csv", ".tsv"]:  
 df = pd.read\_csv(filepath) if ext == ".csv" else pd.read\_csv(filepath, sep='\t')  
 if file\_role not in df.columns:  
 print(f"Warning: '{file\_role}' column is missing in {filepath}. Using the last column.")  
 df[file\_role] = df.iloc[:, -1].astype(str)  
 else:  
 df[file\_role] = df[file\_role].astype(str)  
 return df  
 elif ext == ".pdf":  
 text = ""  
 try:  
 with open(filepath, "rb") as f:  
 reader = PyPDF2.PdfReader(f)  
 for page in reader.pages:  
 text += page.extract\_text() + "\n"  
 except Exception as e:  
 messagebox.showerror("File Read Error", f"Error reading PDF file: {e}")  
 return None  
 lines = [line.strip() for line in text.split('\n') if line.strip()]  
 return pd.DataFrame({file\_role: lines})  
 elif ext == ".docx":  
 text = ""  
 try:  
 doc = Document(filepath)  
 for para in doc.paragraphs:  
 text += para.text + "\n"  
 except Exception as e:  
 messagebox.showerror("File Read Error", f"Error reading DOCX file: {e}")  
 return None  
 lines = [line.strip() for line in text.split('\n') if line.strip()]  
 return pd.DataFrame({file\_role: lines})  
 elif ext == ".parquet":  
 try:  
 df = pd.read\_parquet(filepath)  
 except Exception as e:  
 messagebox.showerror("File Read Error", f"Error reading Parquet file: {e}")  
 return None  
 if "source" not in df.columns:  
 print(f"Warning: 'source' column missing in {filepath}. Using the last column as source.")  
 df["source"] = df.iloc[:, -1].astype(str)  
 else:  
 df["source"] = df["source"].astype(str)  
 if "target" not in df.columns:  
 print(f"Warning: 'target' column missing in {filepath}. Using the last column as target.")  
 df["target"] = df.iloc[:, -1].astype(str)  
 else:  
 df["target"] = df["target"].astype(str)  
 return df  
 else:  
 messagebox.showerror("Unsupported File", f"File type {ext} is not supported.")  
 return None  
  
previous\_target\_file = None

For postprocessing, we implemented evaluation functions such as cross-entropy loss, BLEU Score, and Cosine Similarity for reference of our predicted translations. Here is our code showing how we implemented these functions:

def calculate\_bleu\_score(reference: str, candidate: str) -> float:  
 reference\_tokens = [reference.split()]  
 candidate\_tokens = candidate.split()  
 return sentence\_bleu(reference\_tokens, candidate\_tokens)  
  
def cosine\_similarity(vec1, vec2):  
 dot = np.dot(vec1, vec2)  
 norm1 = np.linalg.norm(vec1)  
 norm2 = np.linalg.norm(vec2)  
 return dot / (norm1 \* norm2) if norm1 and norm2 else 0.0

For cross-entropy loss, we use this method later in our code to implement it:

# Compute cross-entropy loss using the reference as labels (this does not affect model weights)  
outputs = model(\*\*input\_enc, labels=reference\_enc)  
loss = outputs.loss

# Extract encoder representations for cosine similarity  
try:  
 encoder\_out = model.model.encoder(input\_enc["input\_ids"], attention\_mask=input\_enc["attention\_mask"],  
 output\_hidden\_states=True)  
 input\_embedding = tf.reduce\_mean(encoder\_out.last\_hidden\_state, axis=1).numpy()[0]  
 ref\_encoder\_out = model.model.encoder(reference\_enc, output\_hidden\_states=True)  
 reference\_embedding = tf.reduce\_mean(ref\_encoder\_out.last\_hidden\_state, axis=1).numpy()[0]  
 cos\_sim = cosine\_similarity(input\_embedding, reference\_embedding)  
except Exception as e:  
 print("Error extracting encoder embeddings:", e)  
 cos\_sim = -1.0  
  
return loss.numpy(), bleu, cos\_sim, pred\_text

def evaluate\_sample():  
 input\_text = input\_entry.get("1.0", tk.END).strip()  
 reference\_text = ref\_entry.get("1.0", tk.END).strip()  
 if not input\_text or not reference\_text:  
 result\_label.config(text="Please enter both input and reference texts for evaluation.")  
 return  
 loss, bleu, cos\_sim, pred\_text = run\_evaluation(input\_text, reference\_text)  
 eval\_text = (f"Cross-Entropy Loss: {loss:.4f}\n"  
 f"BLEU Score: {bleu:.4f}\n"  
 f"Cosine Similarity: {cos\_sim:.4f}\n"  
 f"Predicted Translation:\n{pred\_text}")  
 result\_label.config(text=eval\_text)  
  
def on\_evaluate\_button\_click():  
 threading.Thread(target=evaluate\_sample).start()

To compare our model to Google Translate, we need to call Google Translate as an API using Asyncio and then have it translate whatever word or phrase we put in as well as run our Helsinki model:

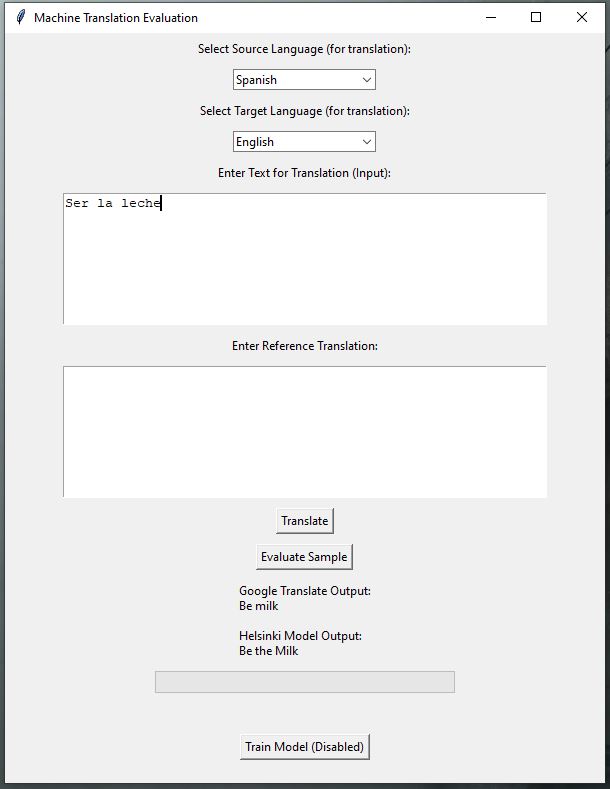
def translate\_text():  
 input\_text = input\_entry.get("1.0", tk.END).strip()  
 if not input\_text:  
 result\_label.config(text="Please enter text to translate.")  
 return  
  
 selected\_source = selected\_language.get()  
 selected\_target = target\_language.get()  
 src\_code = LANGUAGES.get(selected\_source, "auto")  
 tgt\_code = LANGUAGES.get(selected\_target, "en")  
  
 # Google Translate part  
 try:  
 local\_translator = Translator()  
 google\_trans = asyncio.run(  
 local\_translator.translate(input\_text, src=src\_code, dest=tgt\_code)  
 )  
 except Exception as e:  
 result\_label.config(text=f"Error with Google Translate: {e}")  
 return  
  
 google\_translation\_text = google\_trans.text  
  
 # Helsinki translation model part  
 try:  
 helsinki\_translation\_text = helsinki\_translate\_text(input\_text, src\_code, tgt\_code)  
 except Exception as e:  
 helsinki\_translation\_text = f"Error in Helsinki translation: {e}"  
  
 # Form initial result text with both translations  
 result\_text = (f"Google Translate Output:\n{google\_translation\_text}\n\n"  
 f"Helsinki Model Output:\n{helsinki\_translation\_text}")  
  
 # Check for provided reference translation to calculate BLEU score  
 reference\_text = ref\_entry.get("1.0", tk.END).strip()  
 if reference\_text:  
 bleu\_score = calculate\_bleu\_score(reference\_text, helsinki\_translation\_text)  
 result\_text += f"\n\nBLEU Score vs Reference: {bleu\_score:.4f}"  
  
 result\_label.config(text=result\_text)  
  
def on\_translate\_button\_click():  
 threading.Thread(target=translate\_text).start()

We then build our GUI with buttons to act upon our functions with the following code:

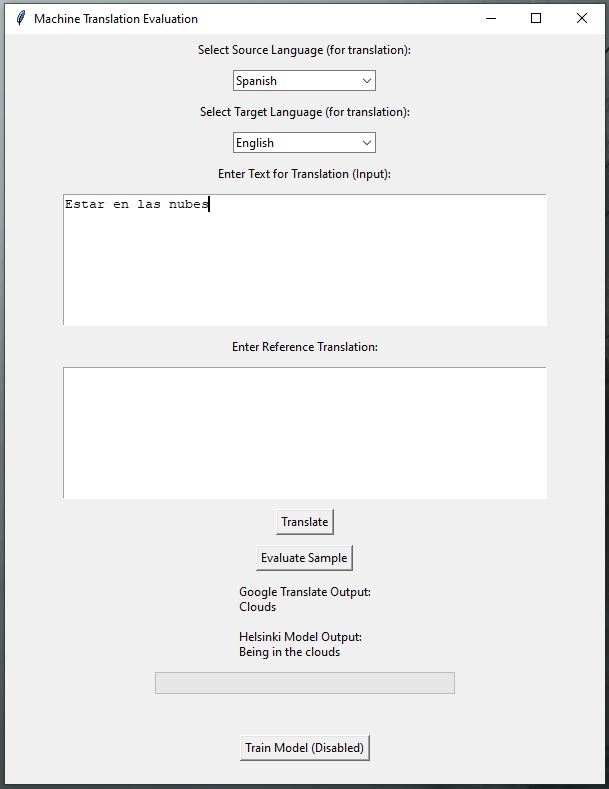
# GUI Setup  
window = tk.Tk()  
window.title("Machine Translation Evaluation")  
window.geometry("600x750")  
  
language\_label = tk.Label(window, text="Select Source Language (for translation):")  
language\_label.pack(pady=5)  
  
sorted\_languages = sorted(LANGUAGES.keys(), key=lambda s: s.lower())  
selected\_language = ttk.Combobox(window, values=sorted\_languages)  
selected\_language.set("Auto Detect")  
selected\_language.pack(pady=5)  
  
# Target Language Options  
target\_language\_label = tk.Label(window, text="Select Target Language (for translation):")  
target\_language\_label.pack(pady=5)  
  
target\_language = ttk.Combobox(window, values=sorted\_languages)  
target\_language.set("English")  
target\_language.pack(pady=5)  
  
input\_label = tk.Label(window, text="Enter Text for Translation (Input):")  
input\_label.pack(pady=5)  
  
input\_entry = tk.Text(window, height=8, width=60)  
input\_entry.pack(pady=5)  
  
# Reference translation  
ref\_label = tk.Label(window, text="Enter Reference Translation:")  
ref\_label.pack(pady=5)  
  
ref\_entry = tk.Text(window, height=8, width=60)  
ref\_entry.pack(pady=5)  
  
translate\_button = tk.Button(window, text="Translate", command=on\_translate\_button\_click)  
translate\_button.pack(pady=5)  
  
evaluate\_button = tk.Button(window, text="Evaluate Sample", command=on\_evaluate\_button\_click)  
evaluate\_button.pack(pady=5)  
  
result\_label = tk.Label(window, text="Results will appear here", justify="left", wraplength=550)  
result\_label.pack(pady=5)  
  
progress\_bar = ttk.Progressbar(window, length=300)  
progress\_bar.pack(pady=5)  
  
training\_info = tk.StringVar()  
training\_info\_label = tk.Label(window, textvariable=training\_info)  
training\_info\_label.pack(pady=5)  
  
# Data loading for fine tuning  
def start\_training():  
 source\_data, target\_data = load\_data()  
 if source\_data is None or target\_data is None:  
 return  
 messagebox.showinfo("Info", "Fine-tuning is disabled in this evaluation setup.")  
  
train\_button = tk.Button(window, text="Train Model",  
 command=lambda: threading.Thread(target=start\_training).start())  
train\_button.pack(pady=5)  
  
window.mainloop()

# Results and INterpretation

The model produced translations that closely match the meaning and context of the source text while maintaining grammatical accuracy and fluency. Utilizing the Helsinki pretrained Transformer model we were able to translate many kinds of words and texts from various languages. By implementing our modifications such as other pre/postprocessing methods with the Helsinki model, this allowed the model to efficiently process longer texts while maintaining high translation accuracy. This allowed us to outperform Google Translation for words such as slang or curse words that Google would not be familiar with. Our model was able to predict words more naturally and accurately where some of these slang and unfamiliar words were not even recognized by Google and it would output the same word without translating it.

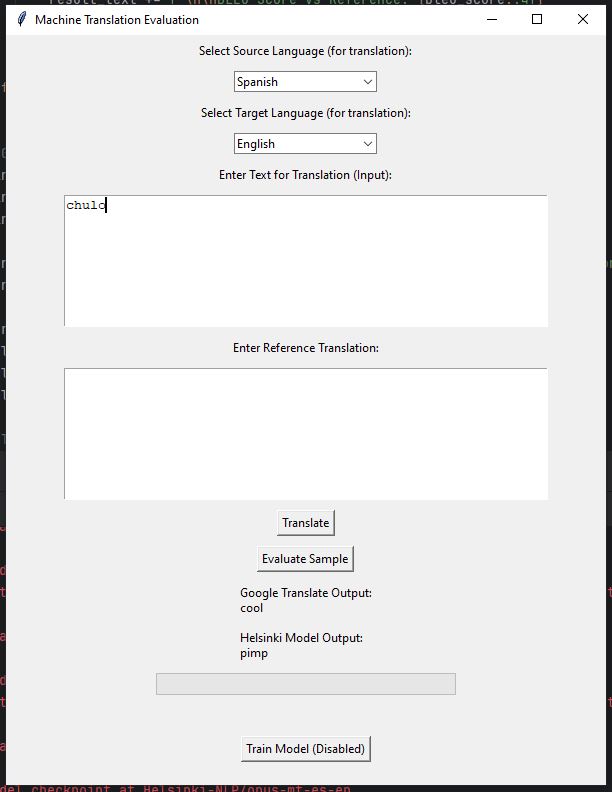


“Ser la leche” was recognized by our model as a more accurate translation compared to Google.

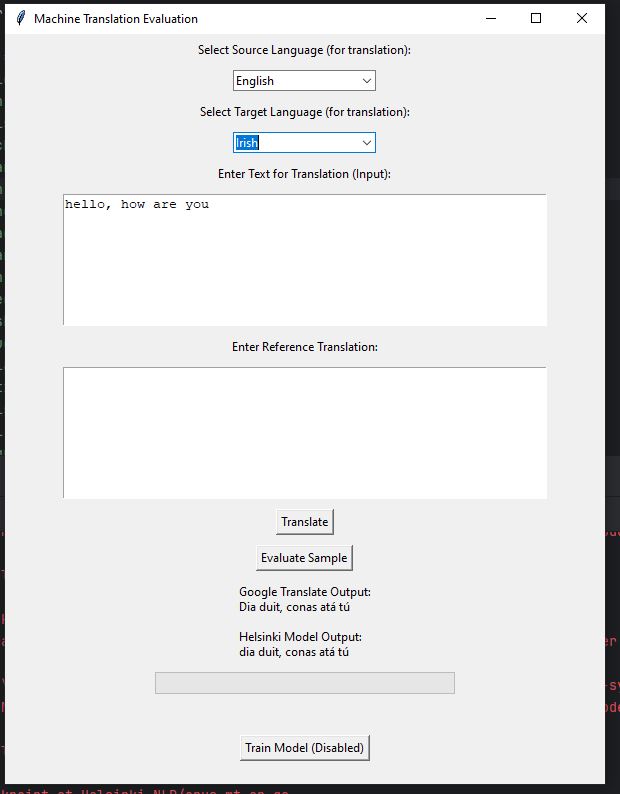


“Estar en las nubes” means to “daydream” and the literal translation is “Being in the clouds” our model recognized this phrase whereas Google only translated it as “Clouds”.

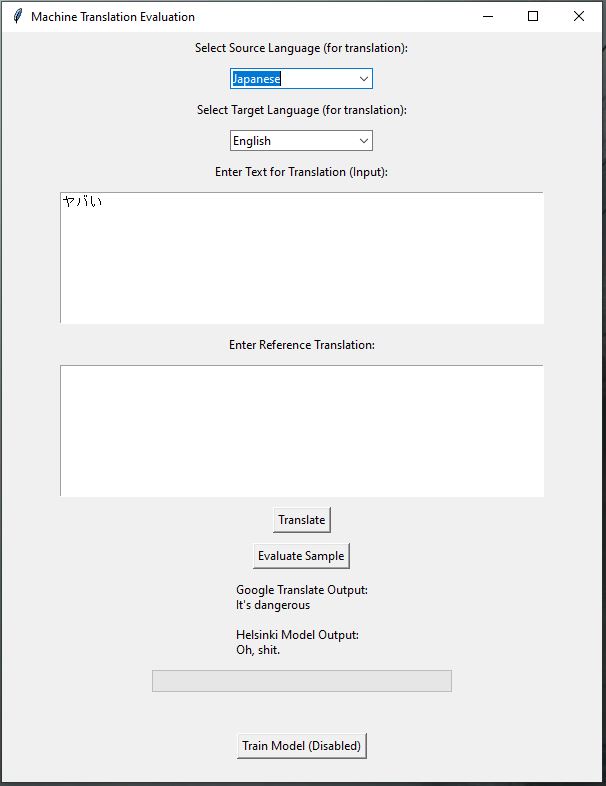
Some words were translated differently by Google compared to the model, but both translations were correct just different such as the following examples:



“chulo” in Spanish shows up as “cool” for Google and “pimp” for our model.

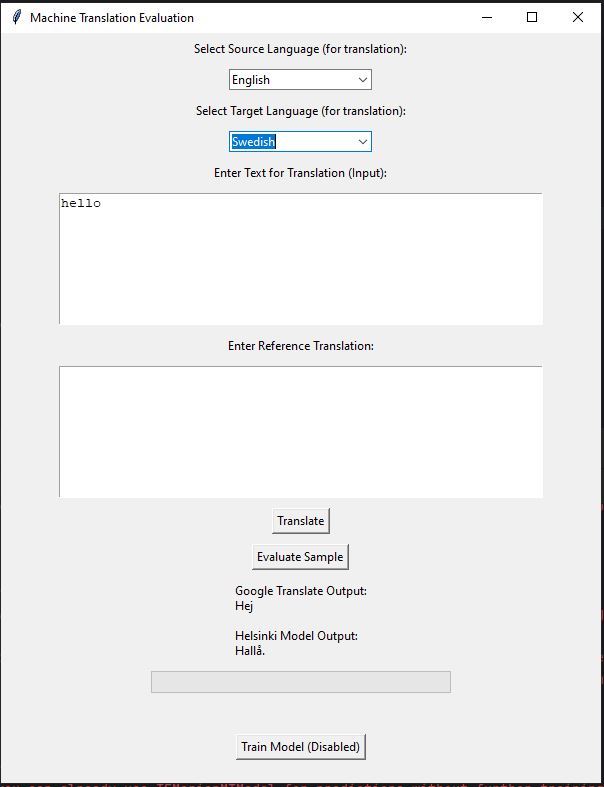


“hello, how are you” in Irish shows up the same, but our model didn’t capitalize it as Google did.

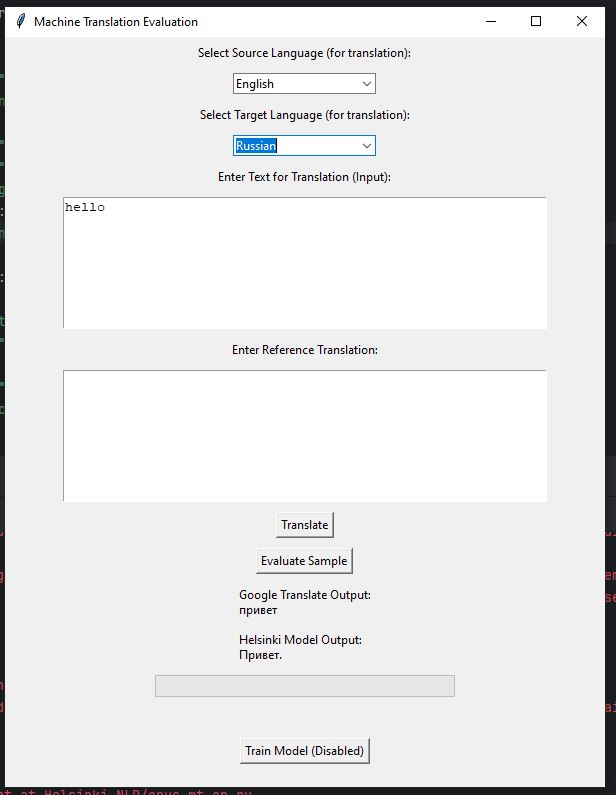


For Japanese the term “yaboi” or “ヤボイ” (the text appears different in our text input, something that needs to be fixed in future iterations for characterization recognization) appears as “It’s dangerous” for Google, and for ours is a more vulgar “Oh, shit.” Both are technically correct depending on the context of the phrase.

Other phrases translated differently, but were minor such as Swedish when saying “hello”, correct punctuation was in outputted from our model unlike Google:

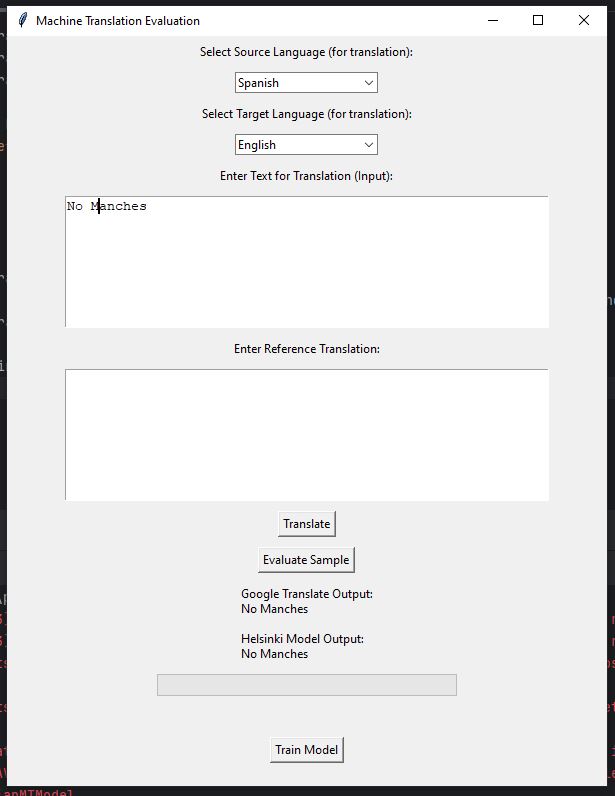


Correct capitalization and punctuation were enabled for various words and phrases such as Russian:



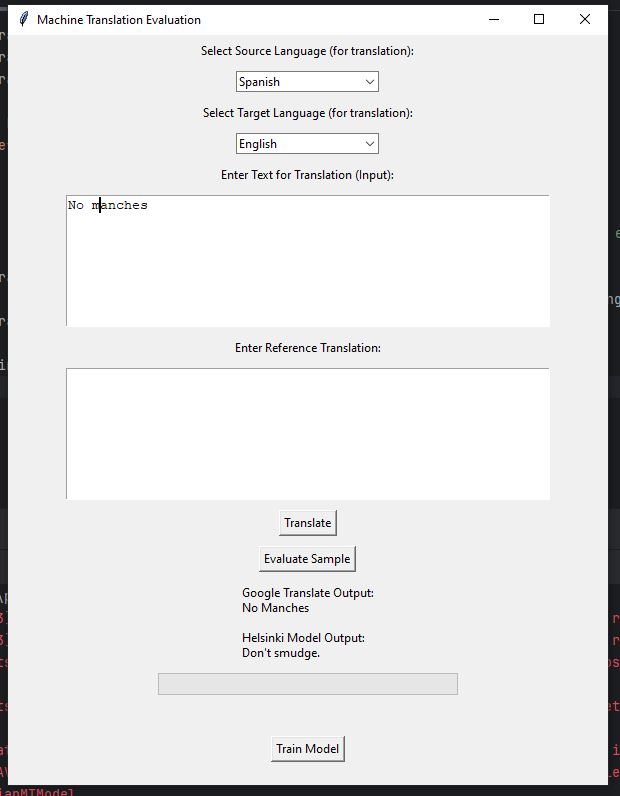
# Discussion of Results

Some limitations of our model are that certain slang or curse words/phrases did not correctly translate to proper English. We also experienced that Welsh did not properly translate at all and needs to be trained much more before it can be implemented in our model. We encountered one phrase that changed depending on the case sensitivity of the letters. For example, the phrase “No manches” translates differently depending on the case of the letters:

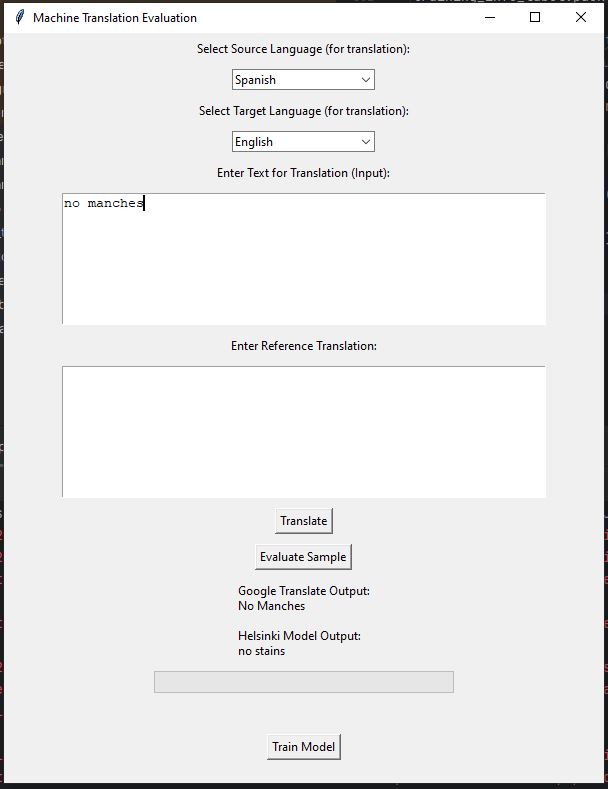


d

Having capital “No” and “Manches” causes our model to not recognize the phrase.



Having lowercase “manches” causes our model to translate it to “Don’t smudge.”



Having all lowercase causes our model to translate it to “no stains”.

# Your Feedback

In the future, there are several improvements we can make to enhance the performance of our model, particularly in handling less common or unfamiliar words during translation. One promising approach would be to integrate an additional model into our system that is specifically trained on datasets containing rare vocabulary, technical words, and idiomatic expressions. By doing so, we can strengthen the model’s ability to recognize and accurately translate these challenging words and phrases, resulting in more precise and natural outputs.

Another potential improvement involves further fine-tuning our current model. Although we adjusted the hyperparameters during our initial development phase, there is still considerable room for optimization. Through more extensive hyperparameter tuning — such as refining the learning rate, adjusting batch sizes, modifying weight decay, and exploring different optimization strategies — we could potentially improve the model’s generalization capabilities. Additionally, implementing targeted fine-tuning on curated datasets that emphasize rare word usage could reinforce the model’s robustness without needing to completely retrain from scratch. We can also implement and test translations of whole paragraphs and stories to make sure that it holds meaning and translations throughout the entire text.

Overall, by combining a specialized secondary model with more aggressive and focused fine-tuning, we believe that future iterations of our translation system will achieve greater accuracy, especially when dealing with less common vocabulary that traditional training might overlook.

Besides the technical aspects, this project was extremely informative and rewarding. Learning about the different components that make up a machine translation system, from data preprocessing and model selection to training strategies and evaluation techniques, gave us a much deeper understanding of how complex this field is. It was exciting to see how each part of the system contributed to the final results, and it motivated us to continue exploring ways to make even more effective and efficient translation models in the future. This project was very informative and a fun way to implement everything we learned in this class!

# References TO DATA SOURCES

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. arXiv preprint arXiv:1409.0473.

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The Long-Document Transformer. arXiv preprint arXiv:2004.05150.

Barrault, L., Bojar, O., Costa-jussà, M. R., Federmann, C., Fishel, M., Graham, Y., ... & Zdrahal, Z. (2019). Findings of the 2019 Conference on Machine Translation (WMT19). Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1), 1-61.

Ferdinan, Teddy, and Jan Kocoń. “Fortifying NLP models against poisoning attacks: The power of personalized prediction architectures.” *Information Fusion*, vol. 114, Feb. 2025, p. 102692, [https](https://doi.org/10.1016/j.inffus.2024.102692)://doi.org/10.1016/j.inffus.2024.102692.

Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The Efficient Transformer. arXiv preprint arXiv:2001.04451.

Koehn, P. (2003). Statistical Machine Translation: Foundations and Recent Advances. The MIT Press.

Lample, G., Denoyer, L., & Ranzato, M. (2018). Unsupervised Machine Translation Using Monolingual Corpora Only. arXiv preprint arXiv:1711.00043.

Naveen, Palanichamy, and Pavel Trojovský. “Overview and challenges of machine translation for contextually appropriate translations.” *iScience*, vol. 27, no. 10, Oct. 2024, p. 110878, <https://doi.org/10.1016/j.isci.2024.110878>.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. arXiv preprint arXiv:1409.3215.

Tiedemann, J. (2012). Parallel Data, Tools and Interfaces in OPUS. Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is All You Need. arXiv preprint arXiv:1706.03762.

Ziemski, M., Junczys-Dowmunt, M., & Pouliquen, B. (2016). The United Nations Parallel Corpus v1.0. Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016).

Torbarina, L., Ferkovic, T., Roguski, L., Mihelcic, V., Sarlija, B., & Kraljevic, Z. (2024). C*hallenges and Opportunities of Using Transformer-Based Multi-Task Learning in NLP Through ML Lifecycle: A Position Paper.*

Supriyono, A., Wibawa, A. P., Suyono, & Kurniawan, F. (2024). *Advancements in natural language processing: Implications, challenges, and future directions.*