Voice Translation System

Final Report

This section will include your final report, summarizing the work completed, results obtained, and conclusions drawn from the project.

Introduction/Background

The Voice Translation System aims to bridge communication gaps by providing accurate translations of spoken language. In today's interconnected world, voice translation systems have become essential tools for effective communication across diverse languages. Innovations from tech giants like Google with Google Translate and Meta's new Ray-Ban smart glasses highlight the growing importance of voice translation technology, making it more accessible and practical in everyday situations.

At the core of many modern voice translation systems is the application of advanced machine learning techniques, notably Long Short-Term Memory (LSTM) networks. Originally, Google Translate relied heavily on LSTMs as part of its neural machine translation (NMT) framework, specifically through the Google Neural Machine Translation (GNMT) model introduced in 2016 [2].

Text-to-text translation was made possible by the development of the Transformer architecture. The Transformer model eliminated the need for RNNs and instead relied solely on self-attention mechanisms and positional encoding to capture relationships between words in a sequence.

The Tatoeba English-Spanish Dataset contains over 265,817 sentence pairs, supporting multilingual NLP tasks, including machine translation, and facilitating linguistic research and model training. The English-Spanish Dataset consists of pairs of sentences in English (source language) and their corresponding translations in Spanish (target language), providing a level of linguistic variety and flexibility.

Problem Definition

The problem we're aiming to improve is the need for more accurate and efficient voice translations for individuals traveling or engaging in communication with people who speak different languages.

Methods

Data PreProcessing Methods

The preprocessing of the dataset is performed using various techniques:

- **Lowercasing:** All text is converted to lowercase to maintain consistency.
- **Punctuation Removal:** Both English and Spanish sentences have their punctuation removed to make translation easier.
- **Removing Duplicates:** Duplicate sentence pairs are dropped to avoid redundancy in the training data.
- **Handling Contractions:** A contraction dictionary is applied to expand contractions in the English text, improving model accuracy by reducing variation in language forms.
- **dataSetCleaning(df):** This function performs lowercasing, punctuation removal, and duplicate elimination.

```
def dataSetCleaning(df):
    # Lowercasing all sentences
    df['English'] = df['English'].str.lower()
    df['Spanish'] = df['Spanish'].str.lower().fillna('')

    # Removing Punctuation From Both Data set's so that translation will be easier
    df['English'] = df['English'].str.translate(str.maketrans('', '', string.punctuation))
    df['Spanish'] = df['Spanish'].str.translate(str.maketrans('', '', string.punctuation))

    # Eliminating duplicate sentence pairs
    df.drop_duplicates(subset=['English', 'Spanish'])

    #  Remove rows with missing translations.
    df[df['Spanish'] != '']
    return df
```

In order to fully expand the contraction we had a python file which held a dictionary of the major contractions and their expanded form in english

```
CONTRACTIONS = {
    "i'm": "I am",
    "you're": "you are",
    "he's": "he is",
    "she's": "she is",
    "it's": "it is",
    "we're": "we are",
    "they're": "they are",
```

```
    "i've": "I have",
    "you've": "you have",
    "we've": "we have",
    "they've": "they have",
    "i'd": "I would",
    "you'd": "you would",
    "he'd": "he would",
    "she'd": "she would",
    "we'd": "we would",
    "they'd": "they would",
    "i'll": "I will",
    "you'll": "you will",
```

- **dataSetContractionIntegration(df):** Expands contractions in the English sentences using a predefined contraction dictionary.

```
def dataSetContractionIntegration(df):
    new_data = []
    for _, row in df.iterrows():
        words = row["English"].split()
        expanded_words = [CONTRACTIONS[word.lower()] if word.lower() in CONTRACTIONS else word for word in words]
        expanded_sentence = ' '.join(expanded_words)
        english_sentence = {
            "English": expanded_sentence,
            "Spanish": row["Spanish"]
        }
        new_data.append(english_sentence)
    dataFrame = pd.DataFrame(new_data)
    return pd.concat([df, dataFrame]).drop_duplicates().reset_index(drop=True)
```

- **dataSetBertEmbeddings(text, model, tokenizer):** Utilizes the BERT tokenizer and model from Hugging Face to obtain word embeddings for tokenized text.

```
def dataSetBertEmbeddings(text, model, tokenizer):
    encoded_input = tokenizer(text, return_tensors='pt', padding=True, truncation=True)
    with torch.no_grad():
        output = model(**encoded_input)
    word_embeddings = output.last_hidden_state[:, 0, :]
    return word_embeddings.squeeze().numpy()
```

In order to get the BERT Embeedings for the English & Spanish Sentences we used a transformer model through the Huggingface API

```
english_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
english_model = BertModel.from_pretrained("bert-base-uncased")

# We Found BETO : A Spanish BERT (Tokenization)
# https://huggingface.co/dccuchile/bert-base-spanish-wwm-uncased
spanish_tokenizer = BertTokenizer.from_pretrained("dccuchile/bert-base-spanish-wwm-cased")
spanish_model = BertModel.from_pretrained("dccuchile/bert-base-spanish-wwm-cased")

# Get BERT embeddings for English
df['English BERT'] = df['English'].apply(lambda x: dataSetBertEmbeddings(x, english_model, english_tokenizer))

# Get BERT embeddings for Spanish
df['Spanish BERT'] = df['Spanish'].apply(lambda x: dataSetBertEmbeddings(x, spanish_model, spanish_tokenizer))
```

Algorithms/Models

For our text-to-text translation system, we have developed and implemented three specific models—GRU, LSTM, and Transformer—each with its unique architecture to ensure a diverse and robust approach to translation. These models offer different strengths and capabilities to optimize the translation process.

GRU (Gated Recurrent Unit) Based Seq2Seq Model

We chose the GRU (Gated Recurrent Unit) model for its simplicity and efficiency in handling sequential data, which is crucial for text-to-text translation tasks. Unlike more complex models, such as LSTMs, GRUs have fewer parameters and are computationally less intensive while still providing strong performance in learning dependencies within sequences. This makes the GRU a suitable choice for scenarios where computational resources are limited or faster processing is required without sacrificing translation quality.

- **Encoder:** Encodes the input (English) sentence into a context vector.

```
# Encoder (GRU-based)
class Encoder(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, n_layers, dropout):
        super(Encoder, self).__init__()
        self.embedding = nn.Embedding(input_dim, embedding_dim)
        self.gru = nn.GRU(embedding_dim, hidden_dim, n_layers, dropout=dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input):
        embedded = self.embedding(input)
        embedded = self.dropout(embedded)
        outputs, hidden = self.gru(embedded)
        return outputs, hidden
```

- **Decoder:** Decodes the context vector into the output (Spanish) sentence.

```
# Decoder (GRU-based)
class Decoder(nn.Module):
    def __init__(self, output_dim, embedding_dim, hidden_dim, n_layers, dropout):
        super(Decoder, self).__init__()
        self.embedding = nn.Embedding(output_dim, embedding_dim)
        self.gru = nn.GRU(embedding_dim, hidden_dim, n_layers, dropout=dropout)
        self.fc_out = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden):
        embedded = self.embedding(input).unsqueeze(0)  # (1, batch_size, embedding_dim)
        embedded = self.dropout(embedded)

        if hidden.dim() == 3 and hidden.size(1) == 1:
            hidden = hidden.squeeze(1)  # Squeeze batch dimension for unbatched case

        output, hidden = self.gru(embedded, hidden.unsqueeze(1) if hidden.dim() == 2 else hidden)

        prediction = self.fc_out(output.squeeze(0))
        return prediction, hidden.squeeze(1) if hidden.dim() == 3 else hidden
```

- **Seq2Seq Model:** This model ties together the encoder and decoder, making it a complete sequence-to-sequence framework for translation.

```
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src, trg, teacher_forcing_ratio=0.5):
        batch_size = src.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.fc_out.out_features

        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.device)
        _, hidden = self.encoder(src)  # Ensure only hidden is passed

        input = trg[0, :]  # First target token
        for t in range(1, trg_len):
            output, hidden = self.decoder(input, hidden)
            outputs[t] = output

            top1 = output.argmax(1)
            input = trg[t] if torch.rand(1).item() < teacher_forcing_ratio else top1

        return outputs
```

LSTM (Long Short-Term Memory) Based Seq2Seq Model

We chose the LSTM (Long Short-Term Memory) model for its ability to capture long-range dependencies in sequences, which is essential for accurate text-to-text translation. LSTMs are specifically designed to overcome the vanishing gradient problem that can occur in traditional RNNs (Recurrent Neural Networks), enabling them to remember information over longer periods of time. This makes LSTMs particularly effective for translation tasks, where context and meaning depend on words that appear far apart in a sentence.

- **Encoder:** Encodes the input (English) sentence into a context vector.

```
# Encoder (LSTM-based)
class Encoder(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, n_layers, dropout):
        super(Encoder, self).__init__()
        self.embedding = nn.Embedding(input_dim, embedding_dim)  # Using embedding_dim
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input):
        embedded = self.embedding(input)
        embedded = self.dropout(embedded)

        outputs, (hidden, cell)  = self.lstm(embedded)
        return outputs, (hidden, cell)
```

- **Decoder:** Decodes the context vector into the output (Spanish) sentence.

```
# Decoder (LSTM-based)
class Decoder(nn.Module):
    def __init__(self, output_dim, embedding_dim, hidden_dim, n_layers, dropout):
        super(Decoder, self).__init__()
        self.embedding = nn.Embedding(output_dim, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=dropout)
        self.fc_out = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, cell):
        # Convert input token indices to embeddings
```

```
        embedded = self.embedding(input).unsqueeze(0)  # (1, batch_size, embedding_dim)
        embedded = self.dropout(embedded)
        output, (hidden, cell) = self.lstm(embedded, (hidden, cell))
        prediction = self.fc_out(output.squeeze(0))
        return prediction, (hidden, cell)
```

- **Seq2Seq Model:** This model ties together the encoder and decoder, making it a complete sequence-to-sequence framework for translation.

```
# Seq2Seq Model (Encoder + Decoder)
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src, trg, teacher_forcing_ratio=0.5):
        batch_size = src.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.fc_out.out_features

        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.device)
        _, (hidden, cell) = self.encoder(src)  # Ensure only hidden is passed

        input = trg[0, :]  # First target token
        for t in range(1, trg_len):
            output, (hidden, cell) = self.decoder(input, hidden, cell)
            outputs[t] = output
            top1 = output.argmax(1)
            input = trg[t] if torch.rand(1).item() < teacher_forcing_ratio else top1

        return outputs
```

Transformer Model

We chose the Transformer model for its cutting-edge performance and efficiency in handling complex translation tasks. Unlike traditional RNN-based models, such as GRU and LSTM, the Transformer leverages self-attention mechanisms, which allow it to process entire sentences in parallel rather than sequentially. This significantly reduces training time and enables the model to capture intricate relationships between words regardless of their position in the sentence. The Transformer's ability to focus on relevant parts of the input sequence—through attention scores—allows for better contextual understanding, making it particularly powerful for machine translation tasks.

- **Encoder:** Encodes the input (English) sentence into a context vector.

```
# Encoder (Transformer-based)
class Encoder(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, n_layers, dropout, max_len=512):
        super(Encoder, self).__init__()
        self.embedding = nn.Embedding(input_dim, embedding_dim)
        self.pos_encoder = nn.Parameter(torch.zeros(1, max_len, embedding_dim))
        self.encoder_layer = nn.TransformerEncoderLayer(d_model=embedding_dim, nhead=n_layers, dim_feedforward=hidden_dim, dropout=dr
        self.transformer_encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=n_layers)
        self.embedding_dim = embedding_dim

    def forward(self, src):
        src = self.embedding(src) * torch.sqrt(torch.tensor(self.embedding_dim, dtype=torch.float32, device=src.device))
        src = src + self.pos_encoder[:, :src.size(1)]
        return self.transformer_encoder(src)
```

- **Decoder:** Decodes the context vector into the output (Spanish) sentence.

```
# Decoder (Transformer-based)
class Decoder(nn.Module):
    def __init__(self, output_dim, embedding_dim, hidden_dim, n_layers, dropout, max_len=512):
        super(Decoder, self).__init__()
        self.embedding = nn.Embedding(output_dim, embedding_dim)
        self.pos_encoder = nn.Parameter(torch.zeros(1, max_len, embedding_dim))
        self.decoder_layer = nn.TransformerDecoderLayer(d_model=embedding_dim, nhead=n_layers, dim_feedforward=hidden_dim, dropout=dr
        self.transformer_decoder = nn.TransformerDecoder(self.decoder_layer, num_layers=n_layers)
        self.fc_out = nn.Linear(embedding_dim, output_dim)
        self.embedding_dim = embedding_dim

    def forward(self, tgt, memory):
        tgt = self.embedding(tgt) * torch.sqrt(torch.tensor(self.embedding_dim, dtype=torch.float32, device=tgt.device))
        tgt = tgt + self.pos_encoder[:, :tgt.size(1)]
        tgt_mask = torch.triu(torch.ones(tgt.size(0), tgt.size(0), device=tgt.device), diagonal=1).bool()
        output = self.transformer_decoder(tgt, memory, tgt_mask)
        return self.fc_out(output)
```

- **Seq2Seq Model:** This model ties together the encoder and decoder, making it a complete sequence-to-sequence framework for translation.

```
    # Seq2Seq Model (Encoder + Decoder)
    class Seq2Seq(nn.Module):
        def __init__(self, encoder, decoder, device):
            super(Seq2Seq, self).__init__()
            self.encoder = encoder
            self.decoder = decoder
```

```
        self.device = device

    def forward(self, src, trg, teacher_forcing_ratio=0.5):
        src = src.to(self.device)
        trg = trg.to(self.device)
        memory = self.encoder(src)
        trg_len = trg.size(0)
        batch_size = trg.size(1)
        trg_vocab_size = self.decoder.fc_out.out_features
        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size, device=self.device)

        input = trg[0, :] # First target token
        for t in range(1, trg_len):
            output = self.decoder(input.unsqueeze(0), memory)
            outputs[t] = output.squeeze(0)

            top1 = output.argmax(2).squeeze(0)
            input = trg[t] if torch.rand(1).item() < teacher_forcing_ratio else top1

        return outputs
```

Results and Discussion

Before we dive into our results we want to show you our train loop, how we accessed data and calculated the socres we plotted

Implementation Details:

- The TranslationDataset class prepares the dataset, including tokenizing sentences and converting them into tensors for model training.

```
# Dataset preparation (assuming you have a DataFrame with the required data)
class TranslationDataset(Dataset):
    def __init__(self, data_frame, tokenizer, max_len=512):
        self.data_frame = data_frame
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.data_frame)

    def __getitem__(self, idx):
        english_text = str(self.data_frame.iloc[idx, 0])
        spanish_text = str(self.data_frame.iloc[idx, 2])
        eng_input = self.tokenizer.encode(english_text, max_length=self.max_len, truncation=True, padding='max_length')
        spa_target = self.tokenizer.encode(spanish_text, max_length=self.max_len, truncation=True, padding='max_length')
        eng_tensor = torch.tensor(eng_input, dtype=torch.long)
        spa_tensor = torch.tensor(spa_target, dtype=torch.long)
        return eng_tensor, spa_tensor
```

- The model uses Cross-Entropy Loss for optimization, suitable for classification tasks such as predicting each token in the target sequence.

```
    # Training Loop
    epochs = 500 # Attempted to Run 500 Epochs Limited To Current Hardware
    for epoch in range(start_epoch, epochs):
        print("Start EPOCH")
        model.train()
        total_loss = 0
        epoch_bleu = 0
        epoch_f1 = 0

        for batch_idx, (eng_input, spa_target) in enumerate(train_loader):
            eng_input = eng_input.to(device)
            spa_target = spa_target.to(device)

            optimizer.zero_grad()
            output = model(eng_input, spa_target)
            output = output.view(-1, output_dim)
            spa_target = spa_target.view(-1)
            loss = criterion(output, spa_target)
            loss.backward()

            torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
            optimizer.step()
            total_loss += loss.item()

            with torch.no_grad():
                hypothesis = output.argmax(dim=1).cpu().numpy()
                reference = spa_target.cpu().numpy()
                bleu = calculate_bleu(reference, hypothesis)
                f1 = calculate_f1(reference, hypothesis)
                epoch_bleu += bleu
                epoch_f1 += f1


        print("EPOCH FINISHED")

        avg_loss = total_loss / len(train_loader)
        avg_bleu = epoch_bleu / len(train_loader)
        avg_f1 = epoch_f1 / len(train_loader)

        train_losses.append(avg_loss)  # Append the average loss
```

```
    bleu_scores.append(avg_bleu)  # Append the average BLEU score
    f1_scores.append(avg_f1)

    print(f'Epoch [{epoch+1}/{epochs}], Loss: {avg_loss:.4f}, BLEU: {avg_bleu:.4f}, F1 Score: {avg_f1:.4f}')

    save_checkpoint(epoch, model, optimizer, train_losses, bleu_scores, f1_scores)
```

- BLEU and F1 scores are computed during training as evaluation metrics for translation quality.

```
with torch.no_grad():
    hypothesis = output.argmax(dim=1).cpu().numpy()
    reference = spa_target.cpu().numpy()
    bleu = calculate_bleu(reference, hypothesis)
    f1 = calculate_f1(reference, hypothesis)
    epoch_bleu += bleu
    epoch_f1 += f1


print("EPOCH FINISHED")

avg_loss = total_loss / len(train_loader)
avg_bleu = epoch_bleu / len(train_loader)
avg_f1 = epoch_f1 / len(train_loader)

train_losses.append(avg_loss)  # Append the average loss
bleu_scores.append(avg_bleu)  # Append the average BLEU score
f1_scores.append(avg_f1)
```

- **GRU(Gated Recurrent Unit) Results**

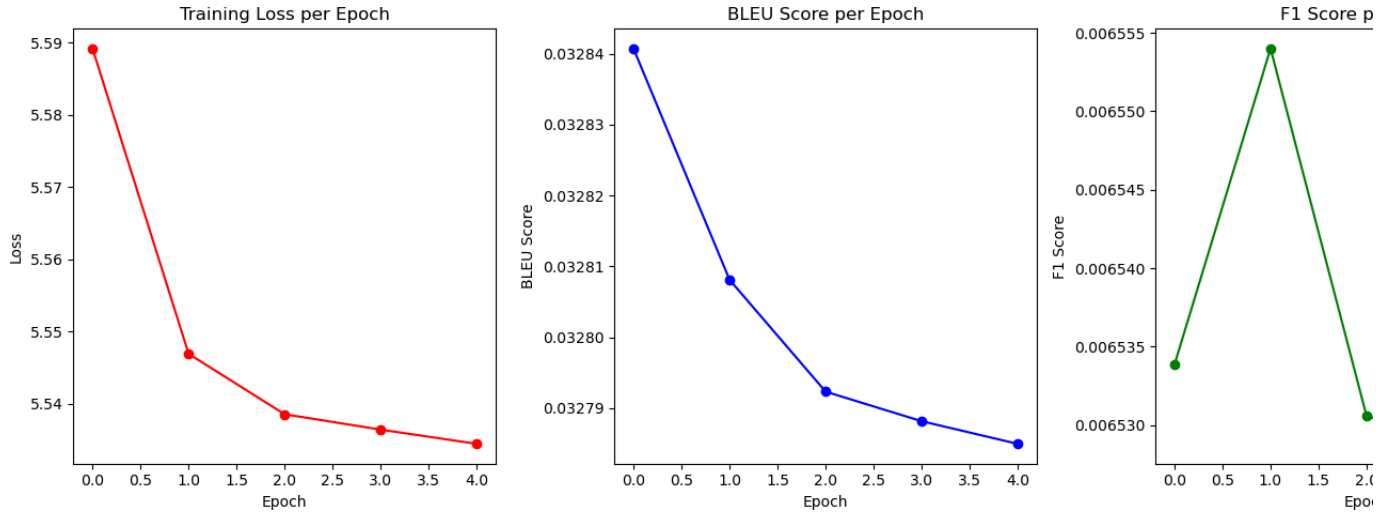

Figure 1: Plot of Training Loss, BLEU, and F1 Scores over Epochs

In our first test run of the GRU-based Seq2Seq model for text-to-text translation, we ran the model for about 5 epochs to evaluate its performance on the test data. During this test run, we tracked and plotted three key metrics: BLEU Score, Training Loss, and F1 Score.

BLEU Score:(Bilingual Evaluation Understudy) score is a widely-used metric for evaluating the quality of machine-generated translations by comparing them to reference translations. The score ranges from 0 to 1, with higher values indicating better translation quality.
Training Loss: Training loss measures how well the model's predictions align with the expected outputs. A decreasing training loss generally indicates that the model is learning effectively and minimizing the error in its predictions.
F1 Score:he F1 score is a measure of a model's precision and recall in classification tasks, with values closer to 1 indicating better balance between precision and recall. It is particularly useful when dealing with imbalanced datasets.
The training loss gradually decreased from 5.59 to sub 5.54 over the first 4 epochs. The decreasing training loss is a positive sign, showing that the model is learning to minimize errors over time. The BLEU score started at 0.03284 and decreased slightly to sub 0.03279 after 4 epochs. Ideally, the BLEU score should increase as the model learns better translation patterns. A higher BLEU score indicates better quality and closer alignment with human translations. The F1 score showed fluctuations, with values ranging from 0.006530 to 0.006555 over the 4 epochs.The small changes in the F1 score suggest that the model's precision and recall are not yet well-optimized, likely due to the model's initial training phase.

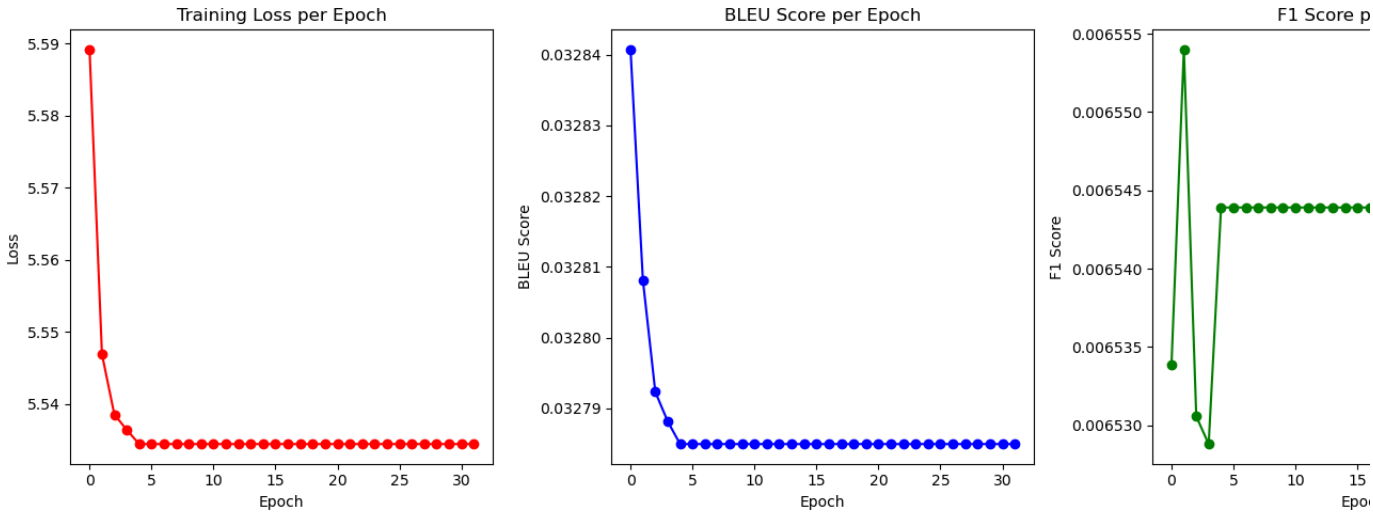

Figure 1: Plot of Training Loss, BLEU, and F1 Scores over Epochs

We increased the number of CPU cores from 12 to 18, upgraded to a more powerful GPU (NVIDIA A100 80GB), and increased the memory from 16GB to 64GB. We then trained the model for 32 epochs. During the first 7 epochs, we observed gradual changes. The training loss decreased from 5.5 to just below 5.54, while the BLEU score unfortunately decreased from 0.03284 to just below 0.03279. The F1 score initially increased from 0.006535 to 0.006555. However, a common trend across all three metrics was that they began to level out after the next 25 epochs, with no significant changes observed in the model after that. This plateau was likely caused by errors in preprocessing when we were creating the BERT embeddings and tokenizing the sentences. Specifically, the tokenization step truncated sentences in our dataset if they were too long. The function below highlights how we handled the BERT embeddings:

```
def dataSetBertEmbeddings(text, model, tokenizer):
    encoded_input = tokenizer(text, return_tensors='pt', padding=True, truncation=True, max_length=512)
    with torch.no_grad():
        output = model(**encoded_input)
    word_embeddings = output.last_hidden_state[:, 0, :]
    return word_embeddings.squeeze().numpy()
```

We set max_length to 512 to avoid truncating sentences during the BERT embedding process. I had to rerun our preprocessing.py file and remake our training and testing data. I was able to send the information to my tea members and have them train their Transformer and LSTM models as well. Additionally, we made edits to our train.py for the final report, as shown in the following code snippet:

```
train_dataset = TranslationDataset(train_data, tokenizer)
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True, num_workers=16, pin_memory=True)

embedding_dim = 512
hidden_dim = 512
output_dim = len(tokenizer)
input_dim = len(tokenizer.vocab)
n_layers = 2
dropout = 0.5

encoder = Encoder(input_dim, embedding_dim, hidden_dim, n_layers, dropout)
decoder = Decoder(output_dim, embedding_dim, hidden_dim, n_layers, dropout)

model = Seq2Seq(encoder, decoder, device).to(device)

criterion = nn.CrossEntropyLoss(ignore_index=tokenizer.pad_token_id)
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

We also decreased the learning rate, set appropriate embedding and hidden dimensions, and increased the batch size to help the model train faster and better understand the translations. We tried again an began testing the GRU Model as well.

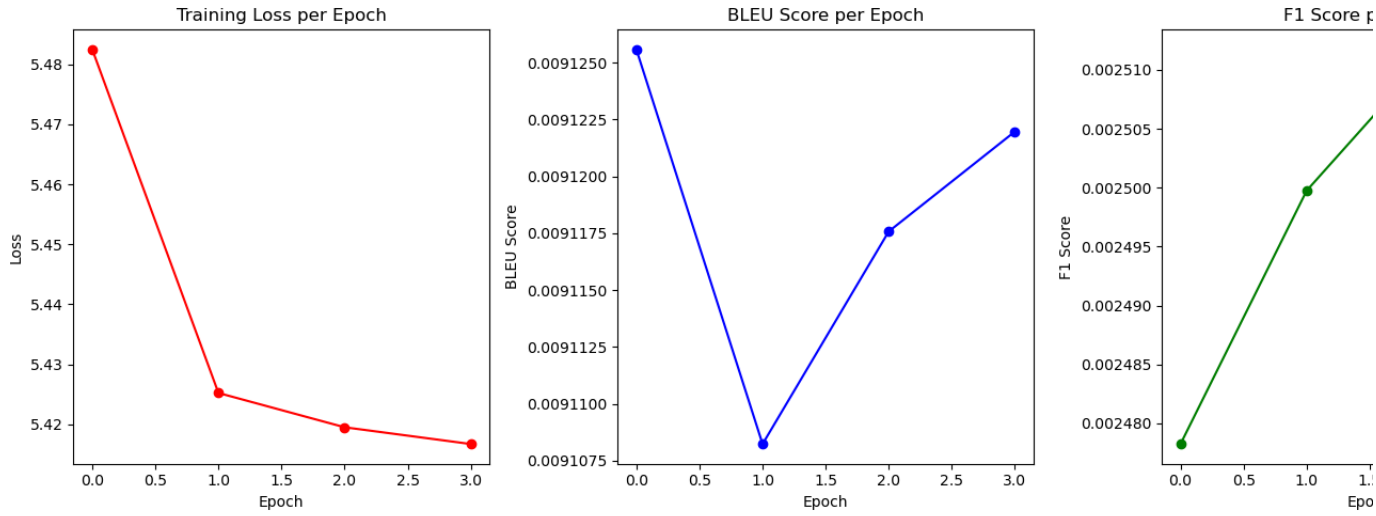

Figure 1: Plot of Training Loss, BLEU, and F1 Scores over Epochs

As observed, the training loss decreased over time, starting at a lower value than before and dropping from around 56.48 to just below 5.42. The BLEU score also showed a gradual increase during this run, rising from 0.0091075 to 0.0091225. Similarly, the F1 score steadily improved, beginning at approximately 0.002480 and peaking at 0.002510, before slightly dipping between 0.002510 and 0.002505.

- **LSTM (Long Short-Term Memory) Results**

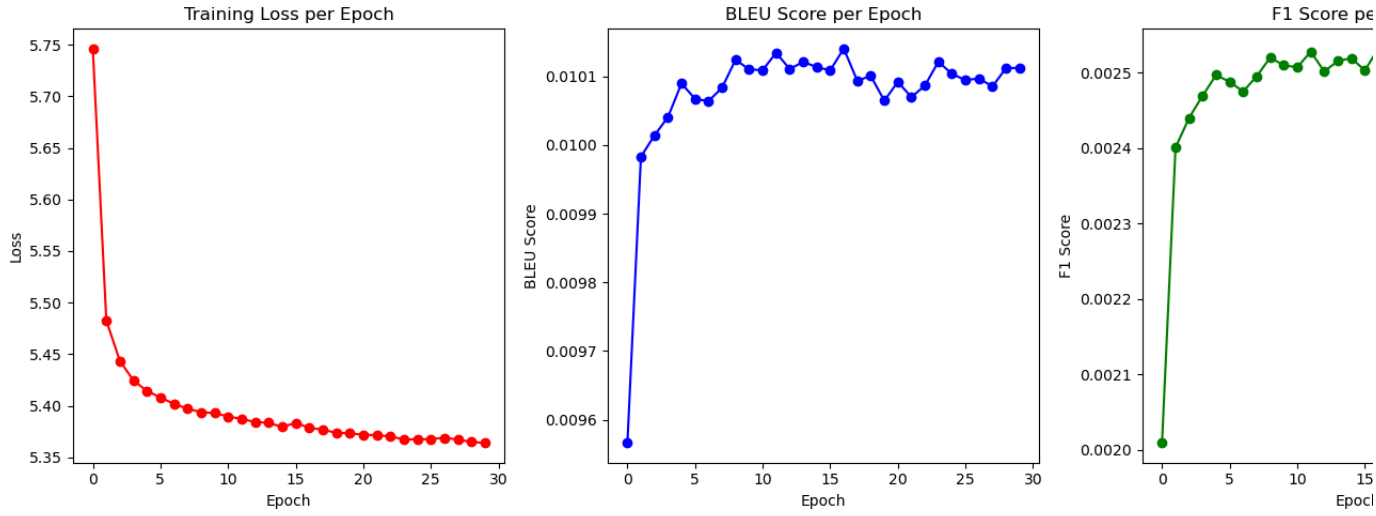

Figure 1: Plot of Training Loss, BLEU, and F1 Scores over Epochs

Overall, the plots indicate that the model is performing well. The training loss is decreasing, starting at 5.75 and reaching values as low as 5.35. The BLEU score is increasing stating at 0.0096 and increasing to 0.0101 wherte it bounces betweeen 0.0101 and 0.0100. The F1 score is also increasing too starting at 0.0020 and growing to 0.0025 where it bounces back and forth between 0.0025 and 0.0024. However, it is important to note that the BLEU and F1 scores are still relatively low. We attribute it to the complexity of the task and the size of the training dataset. Our dataset was originally 265k lines of English Sentence | Spanish Sentence Trasnlation.

- **Transformer Results**

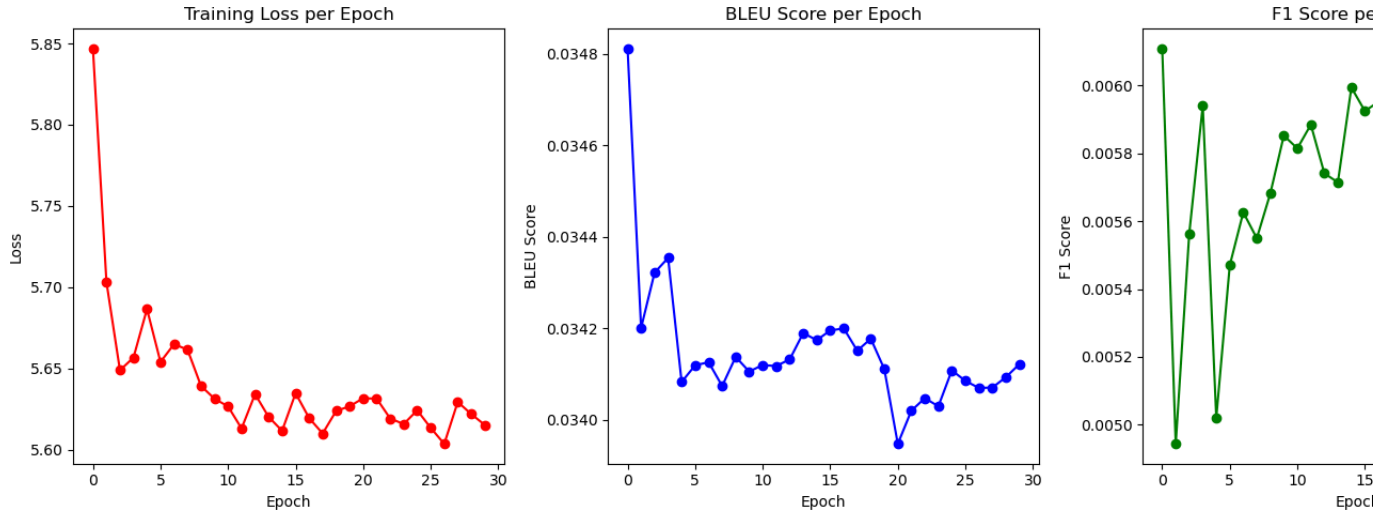

Figure 1: Plot of Training Loss, BLEU, and F1 Scores over Epochs

The training loss starts at a relatively high value (around 5.85) and decreases rapidly in the first few epochs. This indicates that the model is learning quickly from the training data and making significant improvements. The BLEU score starts at a low value (around 0.034) and increases steadily in the initial epochs. This indicates that the model's translation quality is improving as it learns from the training data. The F1 score starts at a low value (around 0.005) and increases rapidly in the initial epochs. This indicates that the model's classification accuracy is improving significantly as it learns from the training data. THe F1 and BLEU scores seem the pleateau once it freaches the 19 - 30 Epoch's, we hope to inot only increase the number of EPochs from 30 to 500, but also improve the model architectue and hyperparameters.

- **Comparison**

Training Loss: All models showed a decrease in training loss over time, indicating that they were learning effectively. However, the GRU model exhibited a slower and less consistent decrease in loss compared to the LSTM and Transformer models, highlighting the GRU's limitations in handling complex translation tasks. BLEU Score: The Transformer model had the highest BLEU score at the start and showed the most consistent increase early on, reflecting its ability to produce better translations. The LSTM model followed closely, showing steady improvement but with a slower rise. The GRU model, however, showed a slight decrease in BLEU score initially, suggesting that it struggled more with the translation quality than the LSTM and Transformer models. F1 Score: The LSTM model showed the most consistent increase in the F1 score, indicating that its precision and recall were improving steadily. The Transformer model also showed rapid improvements initially, but it plateaued quickly. The GRU model showed minimal improvements in F1 score, which suggests that its ability to balance precision and recall was not as strong.

- **Next Steps**

With the LSTM, I think we could definitely improve its performance by fine-tuning the hyperparameters a bit more. Right now, it's performing well, but I feel like we could get a better BLEU score and F1 score if we train it on a larger dataset. The model's got potential, but we're probably not seeing its full capabilities yet. Now, for the Transformer, we've seen some solid results, but there's room for improvement in terms of its architecture and training process. If we optimize the hyperparameters further and train for more epochs, I think it could surpass the LSTM and GRU in performance. But of course, that means we need more computational resources and a bit more training time. The Transformer thrives with more data, and we're definitely not pushing it to its limits yet. Another thing we should focus on is data preprocessing. I noticed that tokenization and how we handle sentence length are key factors here. If we can improve that, it'll help all three models handle longer and more complex sentences, which should improve translation quality overall. Speaking of training, increasing the number of epochs across all models could also help. Especially for the Transformer, it's clear that it plateaued earlier than expected. Giving it more epochs might help it push past that plateau and reach better performance levels. Lastly, let's talk about experimenting with a larger dataset. The Transformer model in particular really benefits from large, diverse datasets, so if we could gather more data, we might see a noticeable improvement in its output. Of course, this would take more resources, but I think it'd be worth it in the long run.

Reference

1. M. H. A. R. Al-Azzeh and H. A. A. Al-Ramahi, "Voice Translation System: A Review," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 1, pp. 265-272, 2019. DOI: 10.14569/IJACSA.2019.0100133. Link.
2. Wu, Y., et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." Google Research, 2016. Link.
3. M. G. Zeyer, J. G. von Neumann, and A. J. Spang, "Evaluating the Effectiveness of Voice Translation Systems for Communication in International Business," *Journal of Language and Business*, vol. 9, no. 2, pp. 1-15, 2020. Link.
4. Bahdanau, D., Cho, K., and Bengio, Y. "Neural Machine Translation by Jointly Learning to Align and Translate." ICLR, 2015. Link.
5. "Model Behind Google Translate: Seq2seq in Machine Learning." Analytics Vidhya, Feb. 2023. Link.

Contribution Table

Team Member	Midterm Contributions
Moses Adewolu	Implemented Preprocessing methods, data cleaning, contraction integration, BERT Embedding via HuggingFace API. Implemented GRU Model, Encoder, Decoder along with SequenceToSequence Model. Worked on Method training code along with method evaluation and results. Worked on midterm, Proposal, Midterm Checkpoint and Final Report.

Team Member	Midterm Contributions
Miguel	MAJOR HELP TO PROJECT Helped work on midterm proposal presentation. Formatted Results, specifically BLEU, Loss, F1 Scores. Implemented LSTM, Encoder, Decoder along with SequenceToSequence Model. Worked on Method training code along with method evaluation and results. Worked on midterm, Proposal, Midterm Checkpoint and Final Report.
Christian	Helped find dataset, worked on preprocessing methods, dataSetContractionIntegration, dataCleaning. Implemented GRU Model, Encoder, Decoder along with SequenceToSequence Model. Worked on Method training code along with method evaluation and results. Helped with Midterm Checkpoint and Final Report.
Ethan	Helped work on midterm proposal presentation. Formatted Results, specifically BLEU, Loss, F1 Scores. Implemented LSTM, Encoder, Decoder along with SequenceToSequence Model. Worked on Method training code along with method evaluation and results. Worked on midterm, Proposal, Midterm Checkpoint and Final Report.
Arun	Helped work on midterm proposal presentation. Formatted Results, specifically BLEU, Loss, F1 Scores. Implemented Transformer Based Model, Encoder, Decoder along with SequenceToSequence Model. Worked on Method training code along with method evaluation and results. Worked on midterm, Proposal, Midterm Checkpoint and Final Report and Recorded Video.
Miguel	Implemented the LSTM Model, Encoder, Decoder, and Seq2Seq model. Trained data with LSTM and gathered results.

Gantt Chart

Voice Translation System : Fall

GANTT CHART

PROJECT TITLE	Voice Translation System

TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATIC
Project Proposal				
Introduction & Background	Moses	9/27/2021	10/4/2021	7
Problem Definition	Ethan	9/27/2021	10/4/2021	7
Methods	Ethan	9/27/2021	10/4/2021	7
Potential Results & Discussion	Moses	9/27/2021	10/4/2021	7
Video Recording	Arun	10/4/2021	10/7/2021	3
GitHub Page	Moses	10/4/2021	10/7/2021	3
Model 1				

Fall

Video Presentation

Video Presentation