Phase 1 – Data Preprocessing and Visualization Report

1. Dataset Overview

The project uses the **Arabic-to-English Translation Sentences Dataset** (Kaggle). It contains matched pairs of english sentences that are used to train and test translation models.

Main Columns:

- **Arabic:** input sentence written in Modern Standard Arabic.
- **English:** corresponding sentence in English.

2. Text Preprocessing

1. Handling Missing Values

- Empty cells were replaced with empty strings to avoid processing errors.

2. Lowercasing

- Both Arabic and English sentences were converted to lowercase for consistency.

3. Punctuation and Number Removal

- Special characters, numbers, and punctuation were removed.

4. Arabic Normalization

- Removal of diacritics (َ ِ ُ ّ)
- Normalizing characters (e.g., "أ/إ/آ" → "ا")
- Removing tatweel (ـ)

5. Tokenization

- Arabic text tokenized using simple whitespace tokenization.
- English tokenized using NLTK word tokenizers.

6. Stopword Removal

- English stopwords were removed for analysis purposes.
- Arabic stopwords removed in exploratory analysis only
  (but kept for translation training to avoid losing meaning).

7. Lemmatization / Stemming

- English lemmatization applied (e.g., “running” → “run”).
- Light Arabic stemming is used carefully (to avoid distorting meaning).

3. Exploratory Data Analysis (EDA)

Basic analysis was conducted to understand the structure and characteristics of the dataset.

3.1 Sentence Length Statistics

- Arabic and English sentence lengths vary from 1–20+ words.
- Most sentences are short, which benefits sequence-to-sequence models.
- English and Arabic sentence lengths showed similar distributions.

3.2 Word Frequency Analysis

Common Arabic words include: أنا، هو، أنت، هذا

Common English words include: **I, you, he, this**

- frequency plots show that the dataset is mainly conversational.

- Frequent verbs and pronouns dominate both languages.

4. Word Representation

To convert text into numeric vectors, the following were applied:

4.1 TF-IDF Features

- Method: Applied TF-IDF (Term Frequency–Inverse Document Frequency) to convert text into numeric features suitable for machine learning models

4.2 Embedding Visualization (PCA)

- PCA (2-D) used to visualize word embeddings.

5. Insights and Observations

- Preprocessing significantly improved text uniformity and reduced noise. ● Sentence lengths show that the dataset is ideal for short-to-medium sequence translation tasks.
- Word frequency analysis reveals that the dataset is primarily conversational.
- TF-IDF convert text into numeric features
- Cleaned and embedded text is now ready for **Phase 2**

Phase 2 Report

Comparison of MiniBERT and LSTM: Model Choice, Architecture, and Training Process

1. Introduction

For this project, two fundamentally different NLP models were selected and implemented: a
Bidirectional Long Short-Term Memory (BiLSTM) network and a MiniBERT model.

2. Model Choice Justification

LSTMs process text sequentially. An LSTM model is lightweight, easy to train, and provides a
meaningful baseline against which modern architectures can be compared.

MiniBERT is a smaller, faster variant of BERT that preserves most of the representational power of large Transformer models while being computationally efficient. MiniBERT employs self-attention, enabling it to capture global contextual information in a single layer. It is pre-trained, multilingual, and optimized for fine-tuning, making well-suited for this dataset containing English and Arabic text.

3. Architecture Comparison

3.1 BiLSTM Architecture

The LSTM model consists of:

1. Embedding Layer
2. BiLSTM Layer
3. Dense + Dropout Layers
4. Output Layer

This architecture has fewer parameters than Transformer models. It learns task-specific
embeddings directly from the dataset and relies on sequential recurrence to process tokens.

3.2 MiniBERT Architecture

MiniBERT follows the Transformer encoder architecture and consists of:

1. Subword Token + Positional Embeddings
2. Stack of Transformer Encoder Layers
3. Classification Head

4. Training Process Comparison

4.1 Data Preparation

Both models use:

● Cleaned English text from Phase 1 as input (the "cleaned" column).

● Labels derived from text characteristics or from provided annotations.

● An 80/20 stratified train/validation split.

4.2 Training the LSTM

● Text is tokenized using a Keras Tokenizer, converted into integer sequences, and
padded to a fixed length.
● Model parameters are learned from scratch.
● Training uses:
○ Higher learning rate
○ More epochs
○ Early stopping and dropout for regularization

4.3 MiniBERT

● Tokenization uses the model’s own AutoTokenizer to produce input ids and attention
mask.
● Because MiniBERT is pretrained, fine-tuning requires:
○ A very small learning rate
○ Few epochs
○ Smaller batch sizes due to GPU memory limits
● Fine-tuning converges quickly because the model already understands grammar, syntax,
semantics, and multilingual patterns.
● Although MiniBERT is heavier per epoch than LSTM, it achieves good performance in
fewer total training iterations.

Phase 3

Bi LSTM

Evaluation Metrics

- Token-level Precision
- Token-level Recall
- Token-level F1-score

Evaluation was performed using greedy decoding, comparing predicted tokens against ground truth tokens (ignoring padding).

```
Token Precision: 0.1003
Token Recall    : 0.0879
Token F1-score  : 0.0937
```

```
print("Example:")
print("EN:", "i like banana")
print("AR:", translate_one("i like banana"))
```

```
Example:
EN: i like banana
AR: احب الرز
```

```
Example:
EN: i love you
AR: انا احبك
```

```python
print("Example:")
print("EN:", "How are you?")
print("AR:", translate_one("How are you?"))
```

```
Example:
EN: How are you?
AR: احبك
```

```python
print("Example:")
print("EN:", "What are you doing?")
print("AR:", translate_one("What are you doing?"))
```

```
Example:
EN: What are you doing?
AR: ماذا تفعل؟
```

Observed Issues

- Wrong translation of some sentences or whole sentences

Causes

- Limited dataset size.
- Greedy decoding.
- No pretrained embeddings were used.

What Worked Well

- Attention mechanism significantly improved translation quality.
- Bidirectional encoder captured richer context.

Challenges Faced

- Handling Arabic morphology.
- Managing special tokens correctly.

Conclusion

This project demonstrates a complete NLP project for machine translation using deep learning despite limitations, the model successfully learns cross-lingual mappings and provides a strong baseline for future improvements

Bert Model

Observed Issues

- Very low BLEU score and poor translation quality in practice (often empty, incomplete, or inaccurate outputs).
- Model appears to misunderstand input sentences in many cases.

Causes

- Input text reconstruction errors (e.g., missing spaces or incorrect casing in English sentences fed to the model).
- No fine-tuning on the specific dataset.
- English-to-Arabic direction is inherently challenging for this model (known weaker performance compared to Arabic-to-English).
- Potential truncation of longer sentences due to default max_length limits.

What Worked Well

- Fast and easy to use.
- Produces natural Arabic on clean input.
- Strong baseline to large pre-training.

Challenges Faced

- Connecting the pre-trained model to our own tokenizer was tricky (needed careful text cleanup and normalization).
- The model often warned about sentences being too long.
- Getting good results required fixing how we prepared the input text.

Conclusion

BERT is a powerful pre-trained baseline for English-to-Arabic translation. Despite the current low observed BLEU due to input preparation issues, it demonstrates the strength of large-scale pre-trained models and significantly outperforms from-scratch approaches when properly integrated.