Assignment 3 : Make Your Own Machine Translation Language

Task 1. Get Language Pair

1. Dataset Selection for Translation Between Native Language (Thai) and English

For the task of training a translation model between Thai and English, I used the dataset from the Hugging Face repository kvush/english_thai_texts, which provides a large collection of aligned English-Thai sentence pairs. This dataset is publicly available and can be accessed from Hugging Face's datasets library.

Dataset Source:

- Dataset: kvush/english_thai_texts
- Repository: Hugging Face (https://huggingface.co/datasets/kvush/english_thai_texts)

2. Process of Preparing the Dataset for Translation Model

The dataset preparation involves several important steps to ensure that the text is in a usable format for training a translation model. Below is a detailed breakdown of the preparation process:

Step 1: Dataset Splitting

The dataset is split into three subsets:

- Training Set: The first 41,901 sentence pairs are used for training.
- Validation Set: The next 5,986 sentence pairs are used for validation.
- Test Set: The last 11,972 sentence pairs are reserved for testing.

This step is performed by selecting specific ranges of rows from the dataset.

Code:

```
train_subset = dataset['train'].select(range(41901))

validation_subset = dataset['train'].select(range(41901, 47887))

test_subset = dataset['train'].select(range(47887, 59859))
```

Step 2: Tokenization

Tokenization is a crucial step in preparing text for neural machine translation. In this case:

- **English Tokenization**: Using the spacy tokenizer for English text. SpaCy is a well-known and efficient library for tokenizing English text. The tokenizer used is the 'en_core_web_sm' model from SpaCy, which is widely used for NLP tasks.
- **Thai Tokenization**: For Thai text, tokenization is a more complex task due to the lack of spaces between words. For this, Using the pythainlp library, which is specifically designed for tokenizing Thai text. The word_tokenize function from PyThaiNLP is utilized to handle the segmentation of Thai sentences.

Code:

```
from torchtext.data.utils import get_tokenizer
from pythainlp.tokenize import word_tokenize token_transform[ENG_LANGUAGE] =
get_tokenizer('spacy', language='en_core_web_sm') token_transform[THAI_LANGUAGE] =
word_tokenize
```

Step 3: Vocabulary Creation

A vocabulary for each language (English and Thai) is created based on the tokenized sentences in the training set. This vocabulary maps each unique token to a corresponding index. Special tokens, such as <unk>, <pad>, <sos>, and <eos>, are also included in the vocabulary. These tokens serve specific roles:

- <unk>: Unknown words (when a token is not found in the vocabulary).
- <pad>: Padding token (for sequences of different lengths).
- <sos>: Start of sequence token (to indicate the beginning of a sentence).
- <eos>: End of sequence token (to indicate the end of a sentence).

The build_vocab_from_iterator function from torchtext is used to generate the vocabulary, ensuring that words appearing less than twice in the training data are treated as <unk> tokens.

Step 4: Text Normalization

In addition to tokenization, text normalization is essential to ensure consistency across the dataset. Common preprocessing includes:

- Lowercasing text (for English).
- Removing special characters or symbols that are irrelevant for translation.
- Stripping extra whitespace from the text to ensure clean tokenization.

Step 5: Sequence Padding and Transformation to Tensors

Since machine learning models typically require inputs to be of the same length, sequences of tokens (i.e., sentences) are padded to the maximum length in a batch. This is handled by the pad_sequence function from PyTorch, which pads sequences with the <pad> token to ensure uniform length across batches.

Moreover, each sentence is transformed into a tensor of token indices (which correspond to the vocabulary indices). Special tokens like <sos> and <eos> are added to the beginning and end of each sentence respectively.

Code:

```
def tensor_transform(token_ids):
```

```
    return torch.cat((torch.tensor([SOS_IDX]),
                      torch.tensor(token_ids),
                      torch.tensor([EOS_IDX])))
```

Step 6: DataLoader Setup

Finally, the prepared data is wrapped into DataLoader objects for efficient batching and shuffling during training, validation, and testing. The collate_batch function is defined to process each batch of data, ensuring that each sample is tokenized, padded, and converted into the appropriate tensor format.

Code:

```
train_loader = DataLoader(train, batch_size=batch_size, shuffle=True, collate_fn=collate_batch)
```

Task 2. Experiment with Attention Mechanisms

1. General Attention: This is the simplest attention mechanism where the attention score is computed as the dot product of the decoder and encoder hidden states.
    - Equation: $e_i = \mathbf{s_t^T h_i}$

    Code:

    # General Attention (dot product)

    energy = torch.matmul(Q, K.permute(0, 1, 3, 2)) / self.scale

2. Multiplicative Attention: In this variation, the query (decoder hidden state) is transformed by a learnable weight matrix W before computing the attention score with the encoder hidden state.
    - Equation: $e_i = \mathbf{s_t^T} \mathrm{W} \mathbf{h_i}$

    Code:

    # Multiplicative Attention (using W)

    q_w = self.W(Q) # Apply transformation W to the query Q

    energy = torch.matmul(q_w, K.permute(0, 1, 3, 2)) / self.scale

3. Additive Attention: In additive attention, the query and key are both transformed by learnable weight matrices, and their results are combined using the tanh activation function. The final attention score is produced by applying a weight vector v.
    - Equation: $e_i = \nu^{\mathbf{T}} \tanh(\mathrm{W}_1 \mathbf{h_i} + \mathrm{W}_2 s_t)$

    Code:

    # Additive Attention (using W1, W2, and v)

    q_w = self.W1(Q) # Transform the query using W1

    k_u = self.W2(K) # Transform the key using W2

    energy = torch.tanh(q_w + k_u) # Apply tanh to the sum of transformed query and key

    attention_scores = self.v(energy).squeeze(-1) # Apply v to get final attention scores

Task 3. Evaluation and Verification

1. **Compare the performance of these attention mechanisms**

Attentions	Training Loss	Training PPL	Validation Loss	Validation PPL
General Attention	1.052	2.863	2.067	7.897
Multiplicative Attention	1.021	2.777	2.018	7.525
Additive Attention	1.108	3.027	2.148	8.565

Translation Accuracy:

- Multiplicative Attention performs the best in terms of translation accuracy with the lowest perplexity (7.580) on the test set.
- General Attention is decent, with a test perplexity of 7.826, showing reasonable accuracy but not as good as Multiplicative.
- Additive Attention has the highest test perplexity (8.489), indicating lower translation accuracy.

Computational Efficiency:

- General Attention is the fastest, taking 0m 50s per epoch, making it the most computationally efficient.
- Multiplicative Attention takes 0m 53s per epoch, slightly slower than General Attention but still relatively efficient.
- Additive Attention takes 0m 59s per epoch, which is the slowest of the three attention mechanisms.

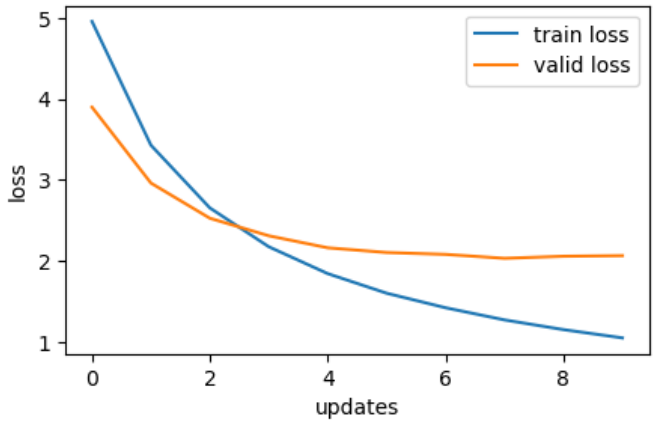
** Running on same device(puffer)

Other Relevant Metrics:

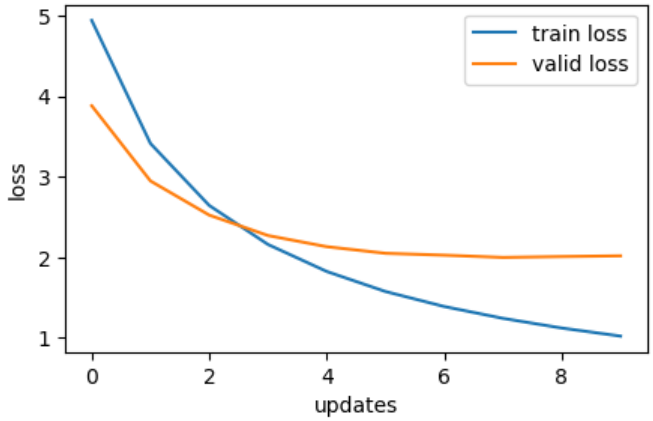
- General Attention strikes the best balance between translation accuracy and computational efficiency. It achieves relatively high performance with faster training time, making it suitable for scenarios requiring quick model iterations.
- Multiplicative Attention, while slightly slower, has a slightly better translation accuracy than Additive Attention, making it a strong candidate when translation quality is prioritized over training speed.
- Additive Attention, although slower and slightly less accurate, still demonstrates reasonable translation performance, though it could be less favorable when computational resources and time are limited.

2. Plots that show training and validation loss for each type of attention mechanism.

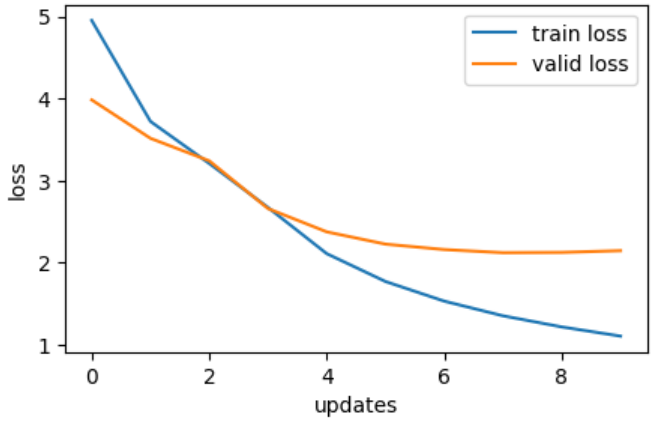
- General attention



- Multiplicative attention



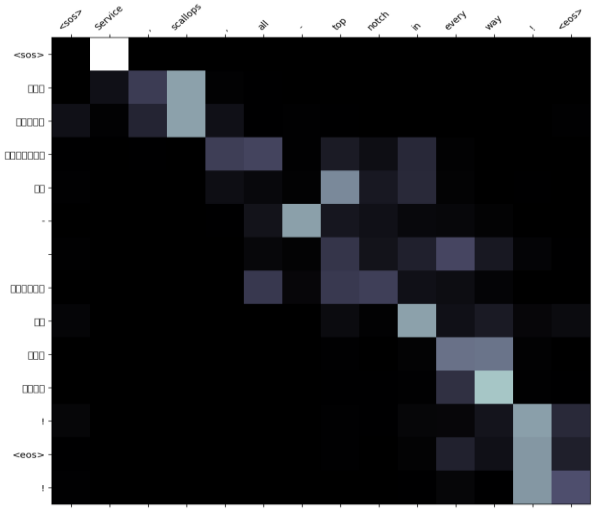
- Additive attention



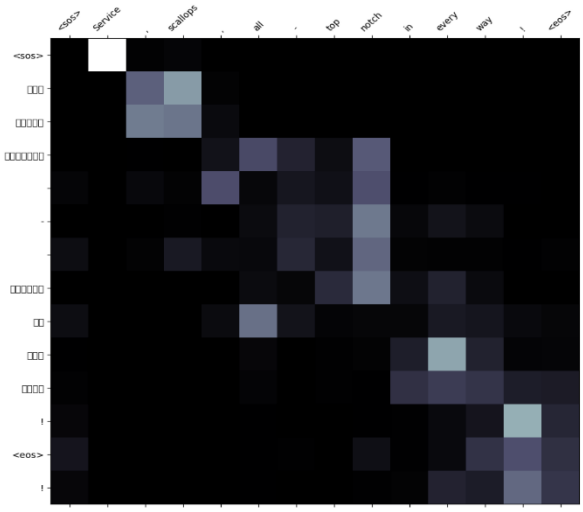
3. Display the attention maps generated by your model

Attention maps are crucial for understanding how a model focuses on different parts of the input sequence when generating a translation, offering transparency and helping with model debugging and improvement.

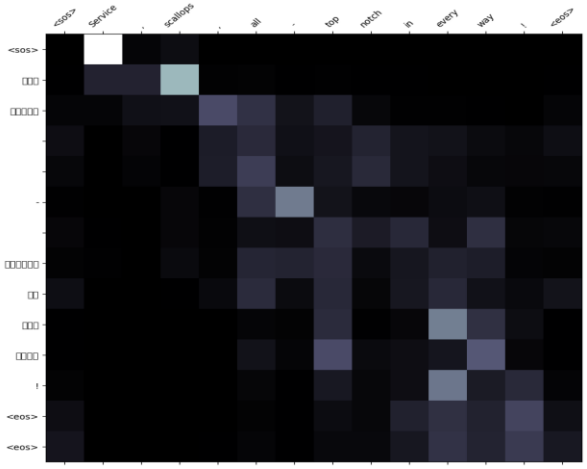
- General attention



- Multiplicative attention



- Additive attention



Visual Representation:

- The attention map is visualized as a heatmap, where each cell corresponds to the attention score between an input token (in the source sequence) and an output token (in the target sequence).
- The attention scores are represented by color intensity: lower values appear darker (black), while higher values appear lighter (ranging from grey to white).

4. Effectiveness of Attention Mechanisms

The effectiveness of attention mechanisms in translating between languages English and Thai, can be evaluated based on translation accuracy, computational efficiency, and attention map interpretability.

- **General Attention** provides the best balance between translation accuracy and computational efficiency. It effectively captures complex dependencies in source and target languages, producing high-quality translations. The attention maps also show varied focus across the input sequence, aligning well with semantic relationships in the translation.
- **Multiplicative Attention** offers strong translation quality by capturing subtle token relationships but is more computationally expensive. This mechanism is ideal for generating accurate translations when resources are not a limiting factor, but it may not be as efficient for large-scale tasks.
- **Additive Attention** is the most computationally efficient, requiring less training time per epoch. However, it sometimes sacrifices translation quality, particularly for complex sentences. It is better suited for simpler tasks or situations where training time is crucial.

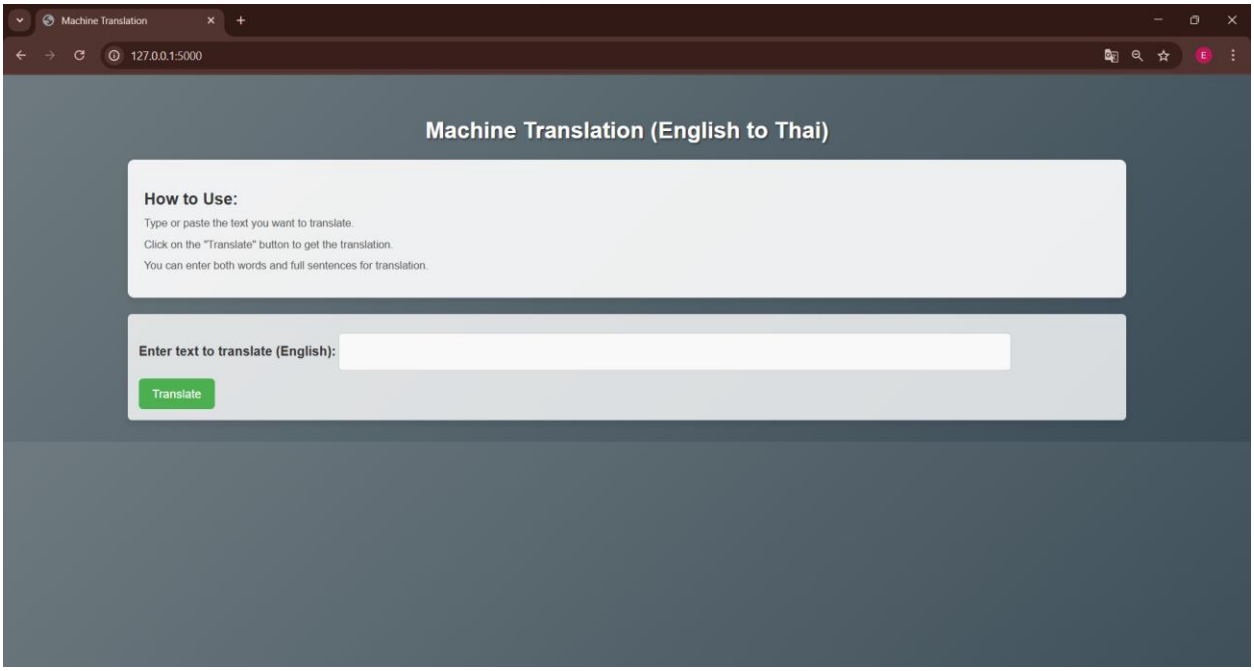
General Attention is the most effective overall, striking a good balance between accuracy and efficiency, making it the optimal choice for translating between languages with different syntactic structures.

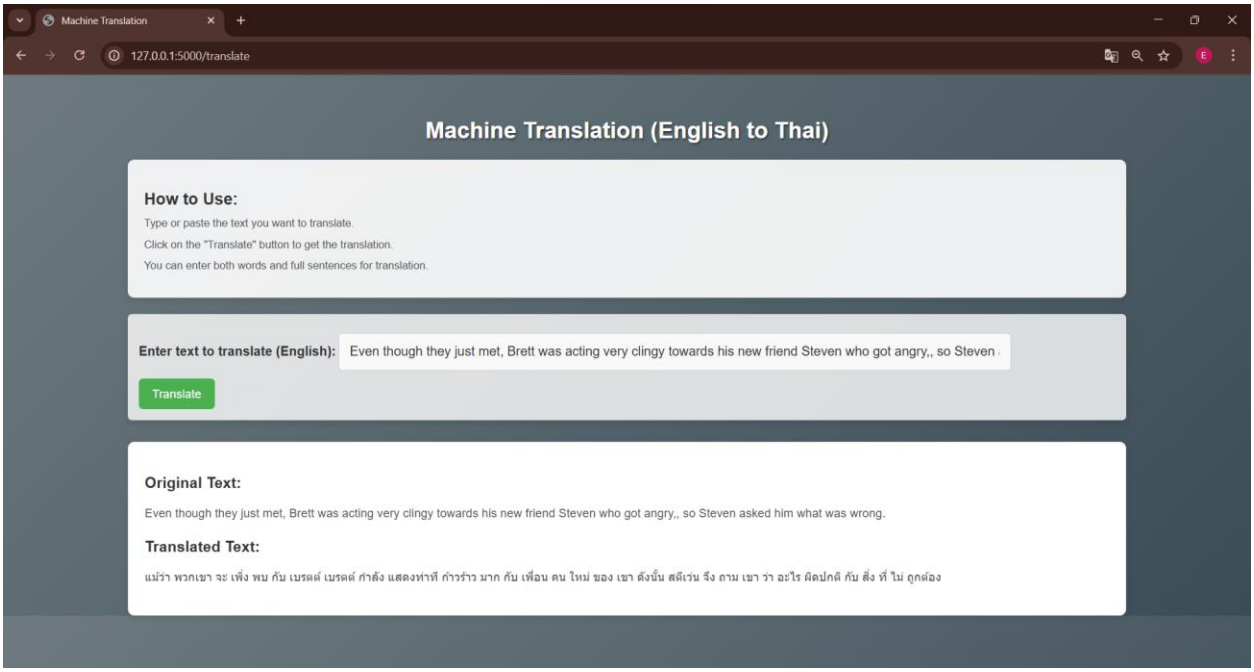
Task 4. Machine Translation - Web Application Development

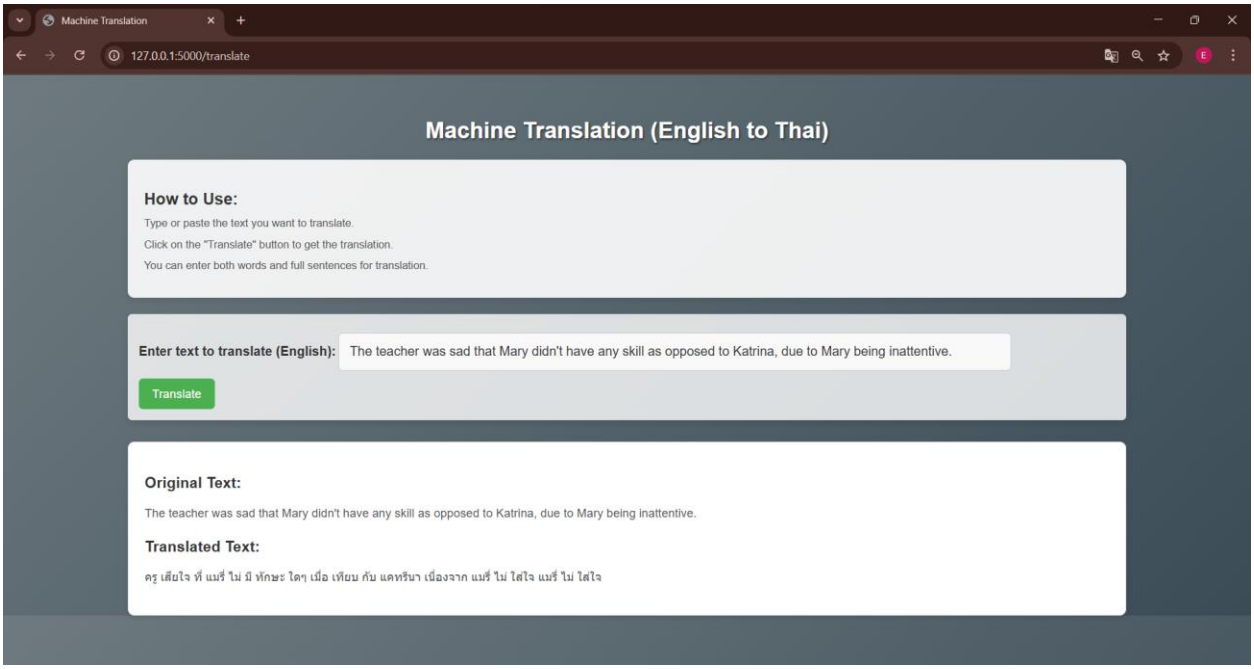
This web application allows users to input text in English and receive a translation in Thai. The translation is generated using a machine translation model.

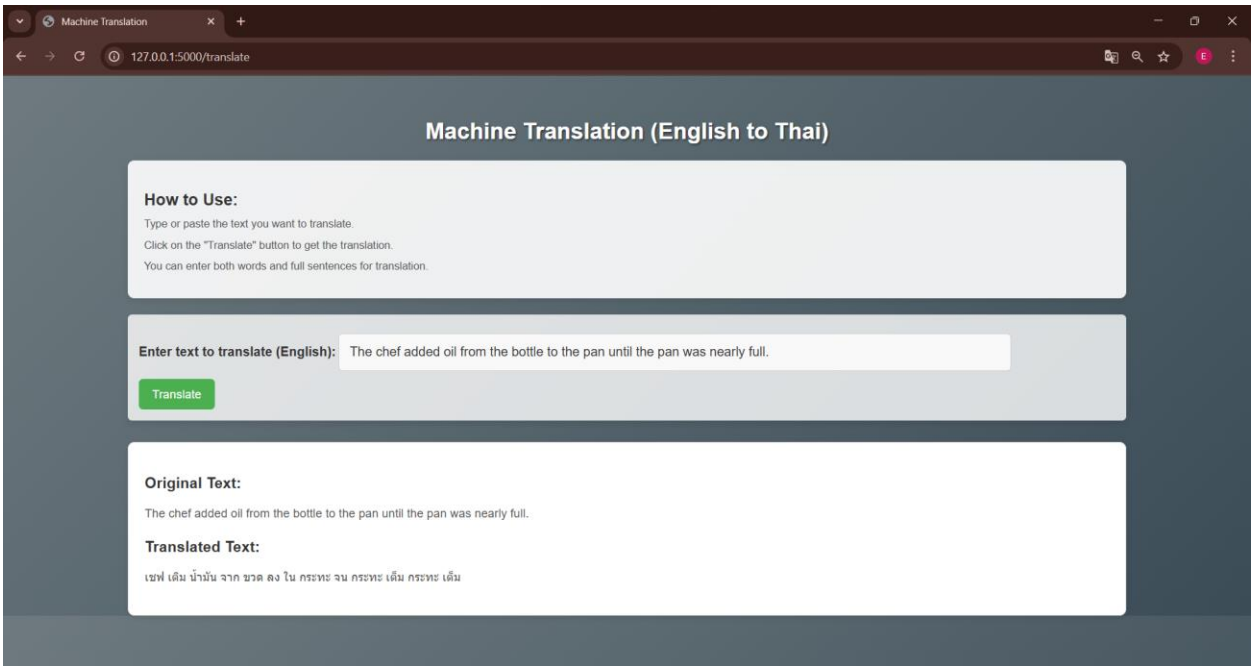
How the App Works:

1. **User Input**: The user types a sentence in English into an input box on the webpage.
2. **Translation Process**:
    - The text is sent to the backend, where a pre-trained machine translation model processes the input.
    - The model translates the sentence from English to Thai.
3. **Output Display**: The translated Thai sentence is then displayed on the webpage


Machine Translation
127.0.0.1:5000
Machine Translation (English to Thai)
How to Use:
Type or paste the text you want to translate.
Click on the "Translate" button to get the translation.
You can enter both words and full sentences for translation.
Enter text to translate (English):
Translate


Machine Translation
127.0.0.1:5000/translate
Machine Translation (English to Thai)
How to Use:
Type or paste the text you want to translate.
Click on the "Translate" button to get the translation.
You can enter both words and full sentences for translation.
Enter text to translate (English): Even though they just met, Brett was acting very clingy towards his new friend Steven who got angry,, so Steven
Translate
Original Text:
Even though they just met, Brett was acting very clingy towards his new friend Steven who got angry,, so Steven asked him what was wrong.
Translated Text:
แม้ว่า พวกเขา จะ เพิ่ง พบ กับ เบรตต์ เบรตต์ กำลัง แสดงท่าที กำวร่ำว มาก กับ เพื่อน คน ใหม่ ของ เขา ดังนั้น สตีเว่น จึง ถาม เขา ว่า อะไร ผิดปกติ กับ สิ่ง ที่ ไม่ ถูกต้อง


Machine Translation
127.0.0.1:5000/translate
Machine Translation (English to Thai)
How to Use:
Type or paste the text you want to translate.
Click on the "Translate" button to get the translation.
You can enter both words and full sentences for translation.
Enter text to translate (English):
The teacher was sad that Mary didn't have any skill as opposed to Katrina, due to Mary being inattentive.
Translate
Original Text:
The teacher was sad that Mary didn't have any skill as opposed to Katrina, due to Mary being inattentive.
Translated Text:
ครู เสียใจ ที่ แมรี่ ไม่ มี ทักษะ ใดๆ เมื่อ เทียบ กับ แคทรินา เนื่องจาก แมรี่ ไม่ ใส่ใจ แมรี่ ไม่ ใส่ใจ


Machine Translation
127.0.0.1:5000/translate
Machine Translation (English to Thai)
How to Use:
Type or paste the text you want to translate.
Click on the "Translate" button to get the translation.
You can enter both words and full sentences for translation.
Enter text to translate (English):
The chef added oil from the bottle to the pan until the pan was nearly full.
Translate
Original Text:
The chef added oil from the bottle to the pan until the pan was nearly full.
Translated Text:
เชฟ เติม น้ำมัน จาก ขวด ลง ใน กระทะ จน กระทะ เต็ม กระทะ เต็ม