VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

NGUYEN HUU THANG – 518H0567
NGUYEN DINH VIET HOANG - 522H0120

FINAL REPORT INTRODUCTION TO NATURAL LANGUAGE PROCESSING

HO CHI MINH CITY, YEAR 2024

VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

NGUYEN HUU THANG – 518H0567
NGUYEN DINH VIET HOANG - 522H0120

FINAL REPORT INTRODUCTION TO NATURAL LANGUAGE PROCESSING

Advised by

Assoc. Prof. Dr. LE ANH CUONG

HO CHI MINH CITY, YEAR 2024

ACKNOWLEDGEMENT

We sincerely thank Assoc. Prof. Dr. Le Anh Cuong for teaching us the Introduction to Natural Language Processing course with great enthusiasm. We want to express our deep appreciation for the dedication and professional knowledge that you shared with us. Through your classes, we gained a better understanding of the fundamental aspects of the Introduction to Natural Language Processing, thanks to your detailed explanations and practical applications. You helped us grasp the knowledge and apply it effectively. Finally, we extend our heartfelt gratitude to Assoc. Prof. Dr. Le Anh Cuong for your commitment and invaluable support throughout our learning journey in this course. The skills and knowledge we acquired will continue to impact our future development. We sincerely thank you and wish your health, success, and happiness.

Ho Chi Minh City, October 28, 2024

Authors:

Nguyen Huu Thang

Nguyen Dinh Viet Hoang

DECLARATION OF AUTHORSHIP

We hereby declare that this thesis was carried out by ourselves under the guidance and supervision of Assoc. Prof. Dr. Le Anh Cuong; and that the work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments, and evaluations from various resources by our own work and have been duly acknowledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknowledged, and explicitly cited.

We will take full responsibility for any fraud detected in my thesis. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

Ho Chi Minh City, October 28, 2024

Authors:

Nguyen Huu Thang

Nguyen Dinh Viet Hoang

ABSTRACT

This document explores the Attention mechanism across various Sequence-to-Sequence architectures, including LSTM-based models, Transformers, and GPT, focusing on their differences and applications in natural language processing (NLP). Attention in LSTM models enhances traditional Encoder-Decoder frameworks by dynamically weighting relevant input states. Transformers revolutionize Attention with Self-Attention and Multi-Head Attention mechanisms, enabling parallel processing and capturing global dependencies. GPT, a Transformer variant, specializes in autoregressive tasks with Causal Attention Masks. Additionally, advanced Transformer-based architectures like RoBERTa and PhoBERT are examined. RoBERTa optimizes training strategies for robust performance on general NLP tasks, while PhoBERT is tailored for Vietnamese, addressing specific linguistic challenges. Comparative analyses, practical demonstrations of Attention in machine translation, and advancements in language modeling establish a comprehensive understanding of Attention's evolution and its transformative impact on NLP applications.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
NER	Named Entity Recognition
Bi-LSTM	Bi-directional Long-Short Term Memory
NLP	Natural Language Processing
NSP	Next Sentence Prediction
MLM	Masked Language Model

EXERCISE 1

1. Theory

a) Describe the Attention mechanism in Sequence-to-Sequence models including:

- LSTM-based model
- Transformer model
- GPT (Generative Pretrain Transformer) model

Analyze the differences in Attention between these architectures.

1. Attention in LSTM-based Models

Sequence-to-Sequence models with LSTM use an Encoder-Decoder architecture:

- **Encoder**: A sequence of LSTMs processes the input $X = (x_1, x_2, \ldots, x_T)$ and compresses the information into a hidden vector h_T or a sequence of vectors $(h_1, h_2, \ldots, h_T)$.
- **Decoder**: Another LSTM sequence uses the hidden vector from the Encoder to generate the output sequence $Y = y_1, y_2, \ldots, y_T$.

Attention Mechanism:

Attention allows the Decoder to reference all hidden states of the Encoder instead of relying solely on h_T.

i. Compute Attention weights:

$$\alpha_{t,i} = softmax(e_{t,i})$$

Where $e_{t,i} = score(s_{t-1}, h_i)$, and s_{t-1} is the Decoder's hidden state at the previous step.

ii. Summarize context:

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$$

c_t is the context vector summarizing information from the Encoder's hidden states.

<u>iii. Incorporate context into the Decoder:</u> The context c_t is fed into the Decoder along with its hidden state s_{t-1} to compute the output.

Key Points:

- Attention allows the Decoder to focus on relevant parts of the input at each decoding step.
- Limitation: Sequential processing makes the model slow and less scalable for long sequences.

2. Attention in Transformer Models

Transformers completely replace LSTM with the Attention mechanism, leveraging an Encoder-Decoder architecture.

Scaled Dot-Product Attention:

Transformers use a more efficient type of Attention called Self-Attention:

<u>i. Attention computation:</u>

$$\text{Attention}(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Q, K, V: Query, Key, Value matrices derived from hidden states.
- $\frac{1}{\sqrt{d_k}}$: Scaling factor to prevent large dot-product values.

<u>ii. Multi-Head Attention:</u> Transformers use multiple attention heads to learn information from different representational spaces.

<u>iii. Placement:</u>

- **Encoder:** Self-Attention captures global context across the input sequence.
- **Decoder:** Uses Self-Attention for the generated sequence and Cross-Attention to integrate Encoder outputs.

Key Points:

- Attention in Transformers eliminates sequential dependence, enabling parallel processing and faster computation.
- Self-Attention captures long-term dependencies in the sequence.

3. Attention in GPT Models

GPT is a Transformer-based architecture using only the **Decoder**.

Attention Mechanism:

GPT relies solely on Self-Attention to model output sequences:

i. Causal Attention Mask: GPT applies a mask to prevent using information from future tokens. This ensures that token t only depends on previous tokens $\{x_1, \dots, x_{t-1}\}$.

ii. Self-Attention: Similar to Transformers, GPT uses Scaled Dot-Product Attention and Multi-Head Attention.

iii. Placement:

- No Cross-Attention, as GPT does not have an Encoder.
- Positional embeddings are added to tokens to encode order information.

Key Points:

- GPT uses Attention to predict the next token (Autoregressive Modeling).
- The main difference from Transformers is the absence of Cross-Attention and reliance on masked Self-Attention.

4. Comparison of Attention Across Architectures

Architecture	**LSTM with Attention**	**Transformer**	**GPT**
Type of Attention	Bahdanau or Luong Attention	Scaled Dot-Product Attention	Scaled Dot-Product Attention
Self-Attention	No	Yes	Yes

Cross-Attention	Yes (Decoder references Encoder)	Yes	No
Sequential	Yes, step-by-step processing	No, parallel processing	No, but with Causal Masking
Applications	Machine translation, small tasks	Machine translation, broader NLP	Text generation, next-token prediction

Table 4: Comparison of Attention Across Architectures

2. Demo

b) Build Lstm-to-Lstm model with and without attention for the problem of Machine Translation from Vietnamese to English:
   - Compare the results between these two models.
   - Show how attention is expressed through an example.

Step 1: Import Libraries

The necessary libraries are imported, including TensorFlow for neural network implementation, NumPy for numerical operations, and Keras tokenization/padding utilities to preprocess text data.

Step 2: Class Definition

The MachineTranslationModel class encapsulates methods for preprocessing, creating models, training, and translating.

2.1 Initialization (__init__)

- Define attributes for:
  - Maximum vocabulary size (max_vocab_size) and sequence length (max_sequence_length).
  - Two tokenizers:
    - vi_tokenizer: For Vietnamese texts.

        - en_tokenizer: For English texts.

Step 3: Preprocessing Data (preprocess_data)

This method processes the input Vietnamese and English texts:

- Fit tokenizers to build vocabulary for both languages.
- Convert texts to sequences of integer token indices.
- Pad sequences to the maximum sequence length for uniform input size.

Step 4: Create Models

4.1 Without Attention (create_lstm_model_without_attention)

- **Encoder**:
    1. Embedding layer to transform token indices into dense vectors.
    2. LSTM layer to produce context vectors (hidden and cell states).
- **Decoder**:
    1. Embedding layer for target language tokens.
    2. LSTM layer initialized with the encoder's states.
    3. Dense layer to map outputs to vocabulary probabilities.

4.2 With Attention (create_lstm_model_with_attention)

- Builds upon the previous model but introduces an **Attention Mechanism**:
    - A custom AttentionLayer calculates:
        1. **Attention scores**: Measure relevance of each encoder hidden state to the current decoder state.
        2. **Attention weights**: Normalize scores using softmax.
        3. **Context vector**: Weighted sum of encoder outputs, dynamically computed for each time step.
    - The decoder incorporates the context vector at each step to improve alignment.

Step 5: Train the Models

5.1 Training Data Preparation

- Input:
  - Encoder input: Processed Vietnamese sequences.
  - Decoder input: English sequences shifted by one token (teacher forcing).
- Output:
  - Target: English sequences shifted by one token (for next-word prediction).

5.2 Training Process

- Both models are trained separately using:
  - Loss function: sparse_categorical_crossentropy for token classification.
  - Metrics: Accuracy to monitor performance.

Step 6: Translate an Example Sentence

6.1 Input Preprocessing

- The Vietnamese input sentence is tokenized and padded.
- A decoder input sequence is initialized with the start-of-sequence token (<start>).

6.2 Translation Loop

- For each time step:
  - Predict the next token based on the current decoder input and encoder output.
  - Append the predicted token to the translated sentence.
  - Stop if the end-of-sequence token (<end>) is generated or the maximum length is reached.

Step 7: Example Usage (main function)

7.1 Prepare Data

- Example Vietnamese and English texts are provided.

7.2 Initialize Class

- Create an instance of MachineTranslationModel.

7.3 Preprocess Texts

- Tokenize and pad the input Vietnamese and English texts.

7.4 Create Models

- Build:
    1. An LSTM-to-LSTM model without attention.
    2. An LSTM-to-LSTM model with attention.

7.5 Train Models

- Train both models using the preprocessed input data.

Step 8: Expected Outputs

- After training, you will obtain two trained models:
    1. Without attention.
    2. With attention.
- Each model can translate Vietnamese sentences into English.
- The model with attention is expected to produce more accurate translations, especially for longer sentences, due to the dynamic focus on relevant parts of the input.

EXERCISE 2

1. Theory

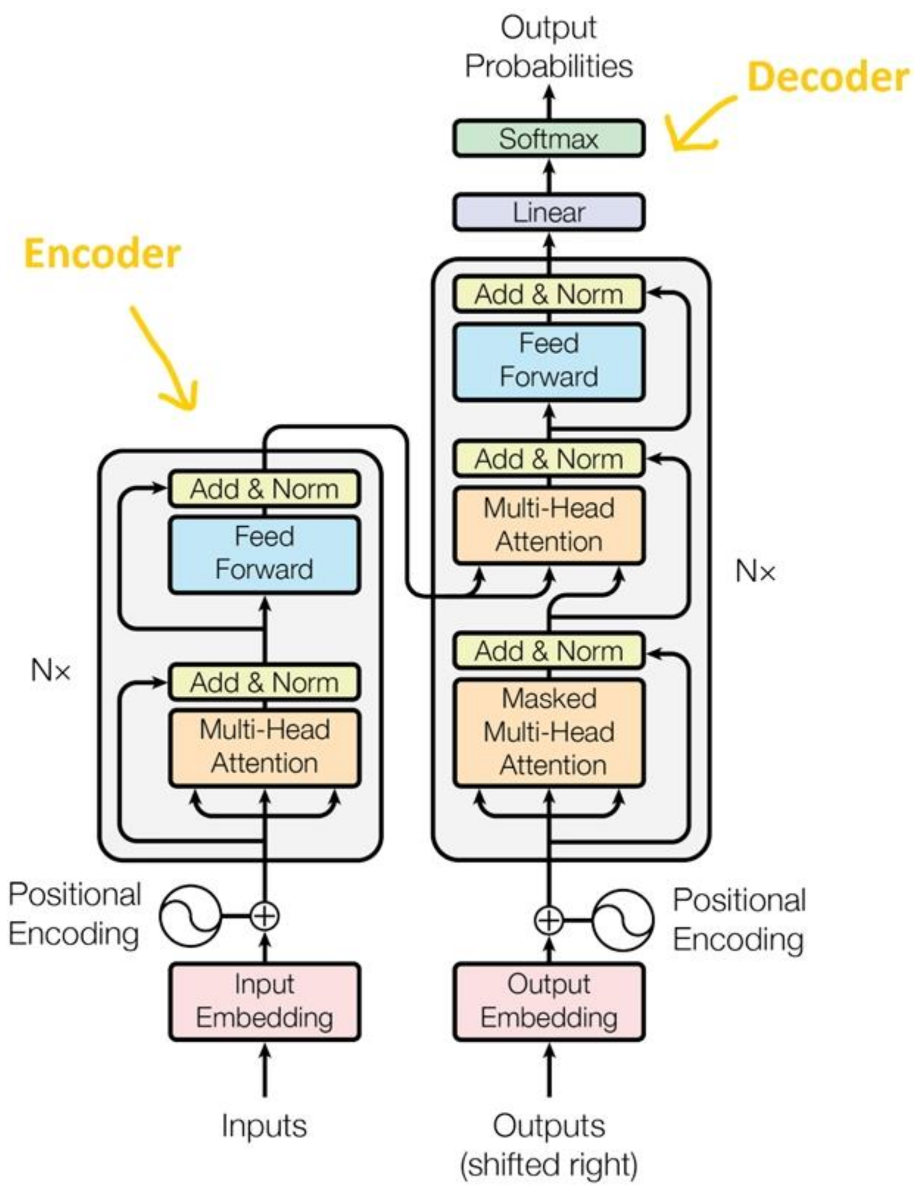


Figure 1: Transformer Architecture

+ Explanation for Figure 1:

- Positional encoding: Since the Transformer does not have recurrent or convolutional layers, it does not inherently know the order of input tokens. Therefore, there needs to be a way for the model to know this information, which is the task of positional encoding. After the embedding layers, which produce token embeddings, we add positional encoding vectors representing the position of each word in the sentence.
- Normalization Layer: In the diagram's architecture, the "Add & Norm" layer refers to the normalization layer. This layer simply normalizes the output of the multi-head attention, improving convergence efficiency.
- Residual Connection: The residual connection is a simple concept of adding the input of a block to its output. This connection allows stacking multiple layers in the network. In the diagram, the residual connection is used after the FFN (Feed-Forward Network) and attention blocks. In the "Add" part of "Add & Norm," it represents the residual connection.
- Feed-Forward Block: This is a basic block where, after performing computations in the attention block at each layer, the next block is the FFN. You can understand that the attention mechanism helps gather information from the input tokens, and the FFN processes that information.

1.1 RoBERTa Model

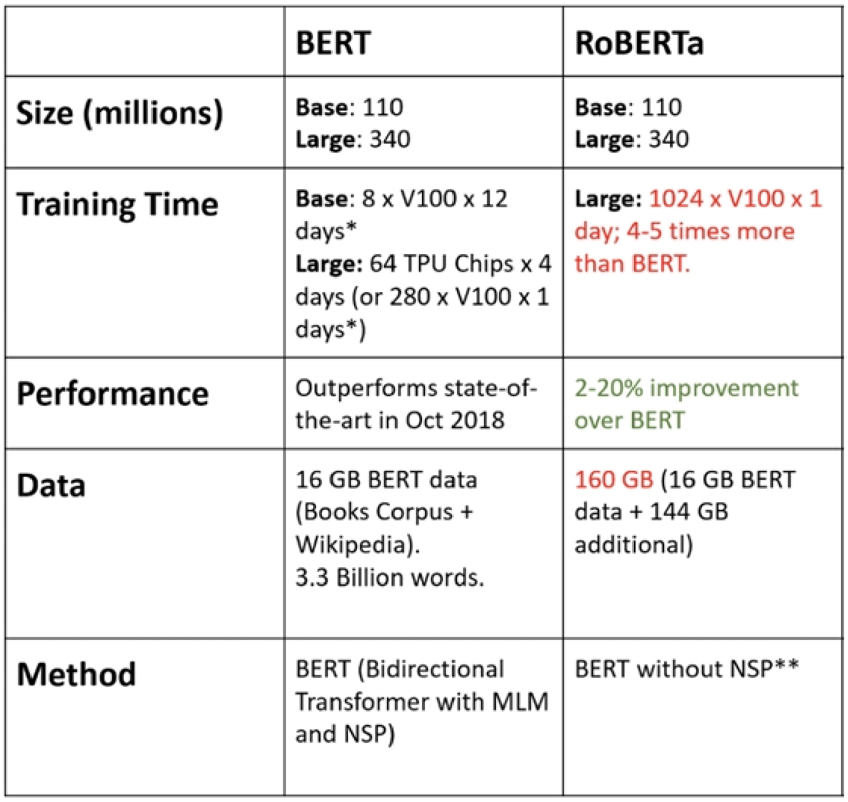
RoBERTa (Robustly Optimized BERT Pretraining Approach) is an improved variant of the BERT model, developed by the Facebook AI Research (FAIR) team. While retaining the Transformer architecture from BERT, RoBERTa optimizes the training process to achieve higher performance on natural language processing (NLP) tasks.

1. Basic Architecture

- **Transformer Encoder**:

- RoBERTa uses the Transformer architecture (Vaswani et al., 2017) with self-attention layers and feed-forward neural networks.
- Two main versions:
  - **RoBERTa-base**: 12 layers, 768 hidden dimensions, 12 attention heads, approximately 125M parameters.
  - **RoBERTa-large**: 24 layers, 1024 hidden dimensions, 16 attention heads, approximately 355M parameters.
- Similar to BERT, RoBERTa is trained to generate contextual embeddings.

2. Key Improvements in RoBERTa



	BERT	RoBERTa
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large:** 1024 x V100 x 1 day; 4-5 times more than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**

Figure 1.1.2: Compare RoBERTa with BERT

RoBERTa does not change BERT's architecture but focuses on optimizing the training process with the following modifications:

- **Increased training data size**:
    - RoBERTa is trained on a larger dataset (~160GB compared to ~16GB for BERT).
    - Sources include: BookCorpus, Wikipedia, Common Crawl News, OpenWebText, and Stories.
- **Modified training objective**:
    - Uses only the **masked language modeling (MLM)** task instead of combining MLM and **next sentence prediction (NSP)** as in BERT. Research shows that NSP does not significantly contribute to model performance.
- **Increased batch size**:
    - RoBERTa uses larger batch sizes (32k tokens compared to 16k tokens for BERT), resulting in more stable training.
- **Longer context**:
    - RoBERTa processes longer contexts in each text sequence, with a maximum sequence length of 512 tokens.
- **Improved tokenization**:
    - Applies **Byte-Pair Encoding (BPE)** with a larger vocabulary (~50k subwords) instead of WordPiece used by BERT.
- **Longer training time**:
    - RoBERTa undergoes more training steps, with careful adjustments to learning rates and warm-up schedules.

3. Performance Improvements

- Thanks to these enhancements, RoBERTa achieves better performance than BERT on many NLP benchmarks such as GLUE, SQuAD, and RACE.
- It is considered a new standard for optimizing pre-trained language models.

4. Applications

- RoBERTa is widely used in NLP tasks such as text classification, machine translation, named entity recognition, and natural language inference.
- Additionally, various derivatives of RoBERTa (e.g., PhoBERT for Vietnamese) have been developed to handle specific languages.

1.2. PhoBERT - The First Large-scale Language Model for Vietnamese (2020)

- **Two versions**: PhoBERTbase and PhoBERTlarge, built on the BERT architecture with enhancements from RoBERTa.

Pre-training Data:

- Utilizes a large Vietnamese dataset (~20GB) comprising Wikipedia (1GB) and Vietnamese news articles (19GB).
- Data is segmented from syllables to words using **RDRSegmenter (Rule-based Discriminative Reranking Segmenter)** and optimized with **Byte-Pair Encoding (BPE)**.

Outstanding Performance:

- Achieves state-of-the-art (SOTA) results on natural language processing (NLP) tasks, including:
  - Part-of-Speech (POS) tagging.
  - Dependency Parsing.
  - Named Entity Recognition (NER).
  - Natural Language Inference (NLI).
- Outperforms multilingual models like **XLM-R**. (XLM-R (XLM-RoBERTa) is a multilingual language model developed by Facebook AI. This model is an extension of RoBERTa and is designed to handle many different languages, including less used ones.)

Applications:

- Provides a robust tool for Vietnamese NLP research and applications.
- Open-sourced on GitHub.

Highlights of PhoBERT:

Addressing Two Major Challenges:

1. The limited and unrepresentative nature of Vietnamese data in Wikipedia.
2. The difficulty of distinguishing syllables from words in Vietnamese.

Effective in Various Vietnamese NLP Tasks:

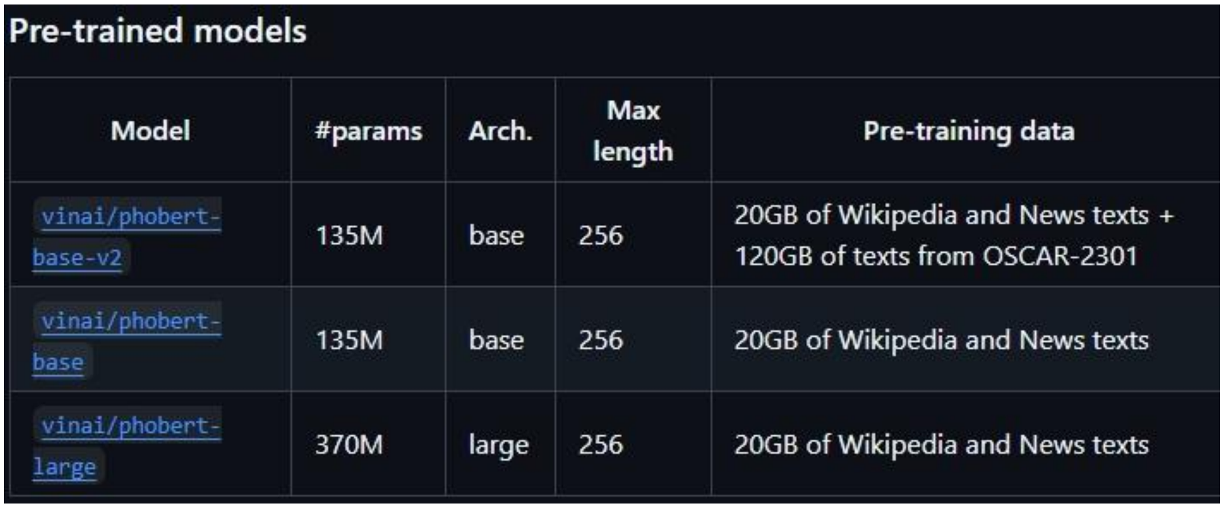
- Significant improvements over existing models in all tested tasks.

Optimized Performance:

- Employs the best training techniques (RoBERTa and BPE).
- Outperforms multilingual models while using fewer parameters.

Development Potential:

- A solid foundation for future research and applications in Vietnamese language processing.



Pre-trained models

Model	#params	Arch.	Max length	Pre-training data
vinai/phobert-base-v2	135M	base	256	20GB of Wikipedia and News texts + 120GB of texts from OSCAR-2301
vinai/phobert-base	135M	base	256	20GB of Wikipedia and News texts
vinai/phobert-large	370M	large	256	20GB of Wikipedia and News texts

Figure 1.2: Different versions of PhoBERT

2. Demo

a) Our group chose a new information extraction problem: extracting information about investments and startups from economic articles.

To build our own training and testing data, our team followed these steps:

Step 1: Define the Datasets

- **Startups**: List of potential startups.
- **Investors**: List of investors.
- **Sectors**: List of investment sectors.

Step 2: Build the Data Generation Function

- **Function generate_investment_description()**:
    - Generate a random sample describing an investment event.
    - Each sample will include:
        - startup (the startup receiving investment).
        - investor (the investor).
        - sector (the sector of the startup).
        - amount (the investment amount).
    - Create one of the investment description templates randomly.

Step 3: Create Training and Test Datasets

- **Generate Training Data (80% of total samples)**:
    - Use a loop to generate around 80% of the total samples for the training set.
    - Each time, call the generate_investment_description() function to create a random sample and add it to the training dataset.
- **Generate Test Data (20% remaining)**:
    - Use a loop to generate 20% of the total samples for the test set.
    - Similarly to above, but only add to the test dataset, not affecting the training set.

Step 4: Save Data to JSON Files

- Save the training data to a separate file named startup_investment_train.json.
- Save the test data to a separate file named startup_investment_test.json.
- Use json.dump() to write the data into the respective files.

b) Build a Transformer model (can use Pretrained model) to train and evaluate the above problem. Note: evaluate on Test set with Precison, Recall, F1-score metrics.

Step 1: Libraries and Modules

Main Libraries:

- **transformers**: Utilized PhoBERT from the Hugging Face library for model training and evaluation.
- **torch**: Handles tensors and creates custom datasets.
- **scikit-learn**: Calculates evaluation metrics such as Precision, Recall, and F1-score.
- **json**: Loads data from JSON files.
- **numpy**: Processes numerical arrays.

Step 2: Custom Dataset Initialization

The StartupInvestmentDataset class is built to prepare data in a format compatible with the model.

2.1. __init__ Method:

- **Input Data**:
  - **data**: Dataset (text sequences and entity labels).
  - **tokenizer**: Tokenizer (PhoBERT tokenizer).
  - **max_len**: Maximum sequence length (128 tokens).
  - **label_encoder**: Converts entity labels to integers.
- **Labels**:
  - Includes: O (Outside), B-... (Begin), I-... (Inside) for each entity type (STARTUP, INVESTOR, SECTOR, AMOUNT).

2.2. __getitem__ Method:

Processes individual samples:

1. **Tokenize Text**: Converts the text into token IDs and pads sequences if they are shorter than max_len.
2. **Initialize Labels**:
    - By default, assigns all tokens the label O (not part of any entity).
3. **Assign BIO Labels**:
    - For each entity in entities, identifies its position in the text.
    - Maps the entity to corresponding tokens and assigns B- (Begin) and I- (Inside) labels.

2.3. __len__ Method:

- Returns the number of samples in the dataset.

Step 3: Performance Metric Computation (compute_metrics)

- **Input**: Prediction results and ground-truth labels.
- **Processing**:
    - Extracts ground-truth and predicted labels, ignoring padding tokens (-100).
    - Calculates metrics: Precision, Recall, and F1-score for each label.
- **Output**: Returns evaluation results.

Step 4: Loading Data from JSON

The load_dataset function reads data from JSON files:

- **Data Format**:
    - Training (train) and testing (test) datasets, each containing:
        - **text**: Text sequence.
        - **entities**: Entities (startup, investor, sector, amount).

Step 5: Main Function - Workflow

5.1. Configuration:

- **PhoBERT Model**: Uses the vinai/phobert-base version.
- **Training Parameters**:
    - Maximum length (max_len): 128.
    - Batch size (batch_size): 16.
    - Number of epochs: 5.
    - Learning rate: 2×10−52 \times 10^{-5}2×10−5.

5.2. Dataset Creation:

- Calls the StartupInvestmentDataset class for training and testing datasets.

5.3. Training Configuration:

- **TrainingArguments**:
    - Defines the number of epochs, batch size, learning rate, and the strategy for saving the best model.
    - Evaluation after every epoch (evaluation_strategy='epoch').
- **Trainer**:
    - Specifies the PhoBERT model, data, and the metrics computation function.

5.4. Training and Evaluation:

1. **Train the Model**:
    - **trainer.train()**: Trains the model on the training dataset.
2. **Evaluate the Model**:
    - **trainer.evaluate()**: Evaluates the model on the test dataset.
    - Prints the Precision, Recall, and F1-score metrics.

REFERENCES

A. Vaswani *et al.*, "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, USA, 2017, pp. 5998-6008.

D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.

T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Lisbon, Portugal, 2015, pp. 1412-1421.

A. Radford *et al.*, "Language Models Are Few-Shot Learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, 2020, pp. 1877-1901.

Y. Liu *et al.*, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," *arXiv preprint arXiv:1907.11692*, 2019.

T. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, Minneapolis, MN, USA, 2019, pp. 4171-4186.

H. T. Nguyen and H. Vu, "PhoBERT: Pre-trained Language Models for Vietnamese," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online, 2020, pp. 1037-1043.

A. Conneau *et al.*, "Unsupervised Cross-lingual Representation Learning at Scale," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online, 2020, pp. 8440-8451.