VIETNAMESE LEGAL TEXT RETRIEVAL

by

Nguyen Hoang Gia Khang; Nguyen Minh Nhat

THE FPT UNIVERSITY

HO CHI MINH CITY

VIETNAMESE LEGAL TEXT RETRIEVAL

by

Nguyen Hoang Gia Khang; Nguyen Minh Nhat

Supervisor: Nguyen Quoc Trung

A final year capstone project submitted in partial fulfillment of the requirement for the Degree of Bachelor of Artificial Intelligent in Computer Science

DEPARTMENT OF ITS
THE FPT UNIVERSITY HO CHI MINH CITY
April 2023 (Month year)

ACKNOWLEDGMENTS

In this section, we would like to acknowledge the following individuals and organizations for their valuable contributions and support:

- Our supervisor, Nguyen Quoc Trung, for his guidance, expertise, and encouragement throughout this project.
- FPT Cloud for reducing fees GPU Server (A30) that greatly aided our research.
- Mr. Kiet Nguyen, for his supporter about UIT-ViQuAD (version 1.0) - A Vietnamese Dataset for Evaluating Machine Reading Comprehension.
- Our friends and family for their love and support throughout our academic journey.

We are grateful to all those who have helped us in ways both big and small, and without whom this project would not have been possible. Thank you all.

AUTHOR CONTRIBUTIONS

Conceptualization, Gia Khang and Minh Nhat; methodology, Gia Khang; software, Gia Khang; validation, Minh Nhat; formal analysis, Gia Khang and Minh Nhat; investigation, Gia Khang; resources, Minh Nhat; data curation, Gia Khang; writing—original draft preparation, Minh Nhat; writing—review and editing, Gia Khang and Minh Nhat; visualization, Gia Khang and Minh Nhat; supervision, Gia Khang; project administration. All authors have read and agreed to the Final Capstone Project document.

ABSTRACT

Vietnamese is a complex language with several nuances, making it challenging to develop effective text retrieval systems. The purpose of this capstone project is to improve Vietnamese text retrieval using the Condenser architecture and Phobert language model. Our objective is to enhance the accuracy and efficiency of existing Vietnamese text retrieval systems, which can have significant implications for search engines, chatbots, and virtual assistants.

We utilized the Condenser architecture and Phobert language model to preprocess the text and extract meaningful features that capture the nuances of the Vietnamese language. We evaluated the performance of our approach using various metrics, including precision, recall, and F2 score, on a Zalo Legal dataset. The dataset contains over 3200 documents with laws.

Based on our results, we conclude that our system has the potential to significantly improve legal research and decision-making, by providing faster and more accurate access to relevant legal texts. Our approach can be adapted to other domains and languages, and has potential applications in fields such as policy analysis and compliance monitoring.

The Condenser architecture [1] combined with the Phobert language model [2] provides a powerful tool for improving Vietnamese text retrieval systems. The proposed approach has the potential to enhance the accuracy and efficiency of text retrieval systems for Vietnamese, which can have significant implications for a wide range of applications, including search engines, chatbots, and virtual assistants. Future work could explore the scalability and generalization of the proposed approach to other languages and domains.

Keywords: Vietnamese text retrieval; Condenser architecture; Phobert language model; precision; recall; F2 score; natural language processing; information retrieval; legal research.

CONTENTS

LIST OF FIGURES

List of Figures

LIST OF TABLES

1 INTRODUCTION

1.1 Overview

1.1.1 Question and Answer System

A Question and Answer (QA) system is a natural language processing (NLP) technology designed to process and respond to user queries or questions in a conversational manner. This system has gained popularity in recent years with the emergence of virtual assistants and chatbots that provide personalized customer support and assistance. QA systems operate by using pre existing data sources or knowledge bases to generate responses to user inquiries. The knowledge base contains information relevant to the specific domain in which the system operates. The system uses natural language processing (NLP) algorithms to understand the question posed by the user and retrieve the most appropriate response from the knowledge base.

The primary goal of a QA system is to provide accurate and relevant responses to user questions in a conversational manner, allowing users to interact with the system in natural language. This technology is becoming increasingly popular in the areas of customer service and support, where users can obtain quick and accurate responses to their queries. QA systems can also be utilized in educational and training applications, where users can ask questions and receive immediate feedback. The system can also be trained to understand multiple languages, making it possible to reach a wider audience.

One of the key challenges in developing QA systems is accurately interpreting user queries or questions. Users may pose questions in a variety of ways and use informal language that may not conform to standard grammatical rules. The system must be able to identify the underlying intent of the question and retrieve the appropriate response from the knowledge base. NLP algorithms are used to analyze the language used by the user and identify the important concepts in the question. The system then uses this information to generate a response that is relevant and accurate. Besides, QA systems are a powerful application of natural language processing technology. These systems have the potential to revolutionize the way we interact with machines and access information. By providing users with a conversational interface and the ability to pose questions in natural language, QA systems are making it easier than ever before to obtain accurate and relevant information. The continued development of natural language processing technology is likely to result in even more sophisticated QA systems in the future, enabling us to engage in increasingly complex conversations with machines.

1.2 Main Topic

QA (Question and Answer) retrieval is a subfield of information retrieval that focuses on finding answers to natural language questions from a large corpus of documents. The goal of QA retrieval is to provide users with concise and relevant answers to their questions, rather than a list of documents that may

or may not contain the answer. QA retrieval techniques often involve natural language processing and machine learning algorithms to analyze the question and retrieve relevant information from the corpus of documents. QA retrieval has numerous applications, such as virtual assistants, customer support, and educational resources.

1.2.1 Vietnamese Law

The behavior of individuals, organizations, and the government in Vietnam is regulated by a complete system known as Vietnamese law. Vietnam's civil law system is heavily influenced by French law and is the foundation of the country's legal system. However, Vietnam's history and culture have also contributed to the creation of its own distinctive legal system.

The Socialist Republic of Vietnam's Constitution is the country's highest legal document and defines the legal system's fundamental principles. The Vietnamese people are the rulers of the country, according to the Constitution, and the state must serve their interests. The socialist-oriented market economy, based on the idea of combining the private sector economy with the state-owned economy, is also recognized by the Constitution.

There are four levels to Vietnam's legal system: the Constitution, regulations, statutes, and announcements. Ordinances and decrees are issued by the government, while laws are passed by the National Assembly. Administrative regulations are also characteristic of Vietnam's legal system, which are issued by government departments and agencies.

The Supreme People's Court, the country's highest court, and the lower courts, which include the provincial and municipal courts, make up Vietnam's legal system. Vietnam's legal system is in charge of interpreting and enforcing the law. The lower courts' decisions can be reviewed and overturned by the Supreme People's Court.

Civil law, criminal law, commercial law, and labor law are just a few of the many areas covered by Vietnamese law. The relationships between individuals and organizations, including marriage, inheritance, and property rights, are governed by civil law. Crimes like murder, theft, and fraud are covered by criminal law. Contracts, intellectual property rights, and bankruptcy are all subjects of commercial law. The relationship between employers and employees is governed by labor law.

The Vietnamese government has worked hard in recent years to reform the legal system and make it easier to understand and use. The framework for the drafting, promulgation, and implementation of laws and regulations was established by the 2017 Law on Promulgation of Legal Documents. All legal documents must be posted online and accessible to the public in accordance with the law.

Additionally, in order to provide low-income individuals and families with legal assistance, the government has established a number of legal aid centers. People who can't afford a lawyer can get free legal advice and representation from these centers. To assist individuals in comprehending their legal rights and

responsibilities, the government has also launched a number of legal education programs.

In conclusion, Vietnam's legal system is a complicated one that has changed over time. The nation's highest legal document, the Constitution lays out the basic principles of the legal system. Civil law, criminal law, commercial law, and labor law are all covered by the legal system. The legal system still faces challenges, such as corruption and a lack of judicial independence, despite the government's efforts to improve transparency and accessibility.

1.2.2 Vietnamese Legal Text Retrieval

Vietnamese legal text retrieval as [6] is a rapidly evolving field that is gaining increasing importance in Vietnam's legal system. Legal text retrieval tools are essential for legal professionals to find, retrieve, and analyze relevant legal documents, including laws, regulations, court decisions, and other legal documents. However, this field presents several significant challenges that require specialized techniques and algorithms to address.

The first major challenge in Vietnamese legal text retrieval is the complexity of the Vietnamese language itself. Vietnamese is a tonal language with a complex grammar structure that can be challenging for non-native speakers to understand. Moreover, legal texts are written in a highly technical language with specific legal terminology that adds another layer of complexity to the task of retrieval. To overcome this challenge, researchers have developed specialized techniques for analyzing Vietnamese legal text, such as natural language processing and machine learning algorithms that can handle the complexity of the language.

Another significant challenge in Vietnamese legal text retrieval is the limited availability of digital legal data. Although Vietnam has made significant progress in digitizing legal documents, there is still a significant amount of legal data that is not available in digital form. This can hinder legal professionals' ability to search for and retrieve relevant legal documents, particularly in cases that require quick access to important legal information. To address this challenge, researchers are developing strategies for converting paper-based legal documents into digital form, such as optical character recognition (OCR) and document scanning technologies.

Despite these challenges, the use of Vietnamese legal text retrieval tools is increasingly important in the fields of legal research and e-discovery. In legal research, Vietnamese legal text retrieval tools allow legal professionals to quickly search through vast amounts of legal data to find relevant information. This can help inform legal strategies and decisions, enabling legal professionals to provide better legal advice and representation. In e-discovery, Vietnamese legal text retrieval tools can help identify, preserve, and collect electronic data for use in legal proceedings.

One of the most promising techniques in Vietnamese legal text retrieval is the use of machine learning algorithms. These algorithms can analyze large amounts of legal text and identify patterns and relationships between legal con-

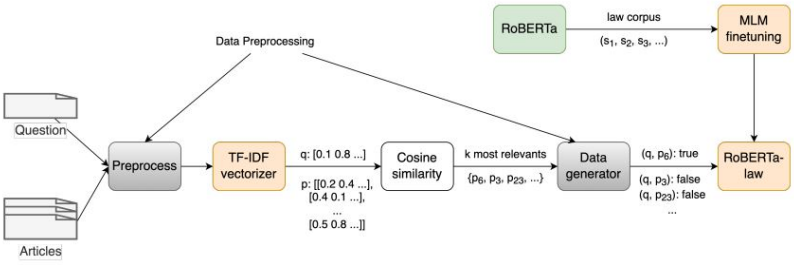


Figure 1: Overview of their Legal Document Retrieval system in [3]

cepts, which can help improve the accuracy and efficiency of legal text retrieval. However, the development of machine learning algorithms for Vietnamese legal text retrieval requires significant resources and expertise.

Despite many challenges, researchers are developing specialized techniques and algorithms for analyzing Vietnamese legal text and retrieving relevant documents. These tools play a crucial role in legal research and e-discovery, enabling legal professionals to quickly search through vast amounts of legal data to find the information they need. As the legal system in Vietnam continues to evolve, the demand for efficient and effective legal text retrieval tools will only continue to grow.

1.3 Specific Works

There are many approaches to approaching and building question-and-answer systems, especially text retrieval systems for legal documents in Vietnamese as deep learning approach [3]. In terms of this approach as the figure 1, The Legal Document Retrieval job seeks to locate all contexts that are, if at all, relevant to a given query given a set of several legal articles (contexts) and a legal inquiry. To keep the most pertinent legal articles, the author used the BM25, a lexical matching model. Their deep learning model can query relevant articles more quickly and for less money thanks to this matching strategy. They also apply this lexical matching technique to generate negative samples as the organizer only offers positive examples. For the purpose of fine-tuning the Roberta model for the text pair classification task, the output of this model is used to generate training samples that include the pair of query sentences, legal articles, and corresponding labels. They finetuned the RoBERTa model [7] by injecting legal linguistic features into a general pre-trained language model in order to improve its effectiveness in a particular domain. Besides, To improve the performance of RoBERTa on the law domain, they finetuning it with 4GB of legal text data which collect this data in two different methods: Gathered straightforwardly from 2 sites "vbpl.vn" and "lawnet.vn". Secoundly, Concentrate sentences near the legitimate point from the news corpus.

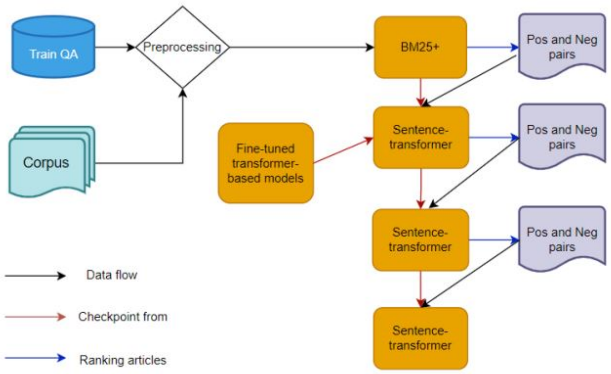


Figure 2: Their proposed pipeline for training in [4]

Other approach as in papper " Multi-stage Information Retrieval for Vietnamese Legal Texts" [4]. Our strategy aims to boost our system's performance by combining lexical matching and semantic searching. For lexical coordinating, we involved BM25+ in bundle rank bm25. For semantic looking, we prepared sentence-transformer models by contrastive learning. The negative sample is 0 and the label of the relevant article to query is 1 for each query in the training dataset. We used the top-k highest ranking score from the previous training round to obtain negative samples. The samples for each query then consisted of positive (pos) and negative (neg) pairs of k+ (number of positive articles). The sentence-transformer model was trained for three rounds using BM25+. Figure 2 depicts our training pipeline.

From both of these approaches, We will combine these two methods by selecting the Phobert language model and crawling data from two reputable legal websites such as "vbpl.vn" and "lawnet.vn". We will go through several steps of data preprocessing to fine-tune the language model for the results of these documents. In the next step, the Condenser architecture will be used to determine the position of the sentence that needs to be queried. To implement this architecture, we will go through four main parts: Finetuning the language model, training the Condenser, training the CoCondenser, and transferring the sentence.

2 RELATED WORKS

2.1 Transformer

2.1.1 Introduction

Definition The Transformer is an architecture for natural language processing tasks that was introduced in the paper "Attention is All You Need" [8]. It is a type of neural network that is designed to process sequential data, such as text, and has achieved state-of-the-art results in a variety of NLP tasks. The Transformer architecture is based on the concept of attention, which allows the network to selectively focus on certain parts of the input sequence when making predictions. Unlike traditional sequence models, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), the Transformer does not rely on recurrence or convolutions to process sequences, which makes it more parallelizable and efficient. The Transformer consists of an encoder and a decoder, each of which contains multiple layers of self-attention and feedforward neural networks. The encoder processes the input sequence, and the decoder generates the output sequence. The self-attention mechanism allows the Transformer to capture long-range dependencies and contextual information in the input sequence, while the feedforward networks enable it to model complex nonlinear relationships between the input and output. Overall, the Transformer architecture has significantly improved the performance of NLP models, especially in tasks that require understanding of long-range dependencies and complex relationships between the input and output.

Importance The Transformer architecture is important because it has revolutionized the field of natural language processing (NLP), enabling significant advances in a wide range of NLP applications. Here are some of the key reasons why the Transformer is important: Improved accuracy: the Transformer architecture has achieved state-of-the-art results in many NLP tasks, surpassing previous models that relied on recurrent neural networks (RNNs) and convolutional neural networks (CNNs). This has led to significant improvements in machine translation, language modeling, text classification, and other NLP applications; better handling of long-range dependencies: The Transformer's attention mechanism allows it to model long-range dependencies between words in a sentence, which is critical for tasks like machine translation where context is important. This is in contrast to RNNs, which have difficulty handling long-term dependencies due to the vanishing gradient problem; more efficient computation: The Transformer architecture is highly parallelizable, which makes it much faster than RNNs and CNNs. This makes it possible to train larger models on larger datasets, which has further improved its performance; flexibility and adaptability: The Transformer architecture is highly modular, which makes it easy to adapt to different tasks and datasets. Researchers have developed many variations of the Transformer architecture for different tasks, such as BERT [9], GPT-2 [10], and T5 [11]

Application Machine translation: The Transformer architecture has been particularly successful in improving the accuracy of machine translation systems. It has enabled the development of models like Google's Neural Machine Translation (GNMT) system, which achieved significant improvements in translation quality. Language modeling: The Transformer architecture has also been used for language modeling tasks, such as predicting the next word in a sentence. Models like GPT-3 [12] have achieved impressive results in this area. Text classification: The Transformer architecture has been used for text classification tasks, such as sentiment analysis and spam detection. It has enabled the development of models that can accurately classify text in real-time. Question answering: The Transformer architecture has been used to develop models that can answer questions based on a given context, such as the Stanford Question Answering Dataset (SQuAD) [13]. Overall, the Transformer architecture has had a profound impact on the field of NLP, enabling significant improvements in accuracy and efficiency in a wide range of applications.

Main components The Transformer architecture consists of several main components, each of which plays a key role in processing sequential data, such as text. Here's a brief overview of the main components of the Transformer:

- Input Embedding: This component converts the input sequence into a high-dimensional vector representation that can be processed by the neural network. Typically, this involves mapping each word or character in the input sequence to a vector using techniques such as word embeddings or character embeddings.
- Positional Encoding: Because the Transformer does not rely on recurrence or convolution, it needs a way to capture the position of each word or character in the input sequence. The positional encoding component adds positional information to the input embeddings, allowing the neural network to differentiate between words or characters based on their position in the sequence.
- Encoder: The encoder component of the Transformer is responsible for processing the input sequence and producing a fixed-length vector representation of the entire sequence. The encoder consists of multiple layers, each of which contains a self-attention mechanism and a feedforward neural network.
- Decoder: The decoder component of the Transformer is responsible for generating the output sequence based on the encoder's fixed-length vector representation of the input sequence. Like the encoder, the decoder consists of multiple layers, each of which contains a self-attention mechanism and a feedforward neural network.
- Self-Attention: The self-attention mechanism is a key component of the Transformer architecture. It allows the neural network to selectively focus

on certain parts of the input or output sequence when making predictions. Self-attention is used in both the encoder and decoder components of the Transformer.

- Multi-Head Attention: The multi-head attention mechanism is a variation of the self-attention mechanism that allows the network to attend to multiple parts of the input or output sequence at the same time. This improves the network's ability to capture complex relationships between different parts of the sequence.

- Output Layer: The output layer is the final component of the Transformer and is responsible for producing the final output sequence based on the fixed-length vector representation generated by the decoder.

Overall, the Transformer architecture's main components work together to enable the network to process sequential data efficiently and accurately. By using self-attention and multi-head attention mechanisms, the Transformer can capture long-range dependencies and complex relationships between different parts of the input and output sequence, leading to significant improvements in NLP tasks.

2.1.2 Attention Mechanism

Attention mechanism definition The attention mechanism is a key component of modern neural networks that allows the network to selectively focus on certain parts of the input when making predictions. Attention mechanisms were first introduced in the context of machine translation to enable the network to selectively focus on different parts of the source sentence when generating the translation. The basic idea behind the attention mechanism is to compute a set of weights that represent the importance of each input element for a given output element. These weights are then used to compute a weighted sum of the input elements, which is used to generate the output. The attention mechanism can be thought of as a way of softening the alignment problem in sequence-to-sequence tasks. Rather than relying on a hard alignment between input and output sequences, the attention mechanism allows the network to learn a soft alignment that can adapt to different input sequences. There are several different types of attention mechanisms, including:

- Scaled Dot-Product Attention: This is a type of attention mechanism used in the Transformer architecture. It computes the dot product of the query vector and the key vectors for each input element, scales the result by the square root of the dimension of the key vectors, and applies a softmax function to compute the weights.

- Additive Attention: This is a type of attention mechanism that uses a feedforward neural network to compute the weights. The input sequence is first transformed using a linear transformation, and then a non-linear function is applied to compute the weights.

- Multiplicative Attention: This is a type of attention mechanism that uses a dot product between the input sequence and a learned weight vector to compute the weights. The dot product is then normalized using a softmax function.

The attention mechanism is a powerful tool that enables neural networks to selectively focus on important parts of the input when making predictions. This has led to significant improvements in a wide range of natural language processing tasks, including machine translation, language modeling, and text classification.

How it works The attention mechanism is a key component of modern neural networks that allows the network to selectively focus on certain parts of the input when making predictions. Here's a detailed explanation of how the attention mechanism works, including its equations and advantages over other methods: Let's say we have an input sequence of length n and an output sequence of length m. The attention mechanism works by computing a set of weights that represent the importance of each input element for a given output element. These weights are then used to compute a weighted sum of the input elements, which is used to generate the output. The basic steps of the attention mechanism can be summarized as follows:

- Compute the Query, Key, and Value vectors: The query, key, and value vectors are computed for each element in the input and output sequences. These vectors are used to compute the attention weights and the weighted sum.Let x be the input sequence of length n, and y be the output sequence of length m. We compute the query vector q_i, key vector k_j, and value vector v_j for each input element x_i and output element y_j as follows:

$$q_i = W_q * x_i \tag{1}$$

$$k_j = W_k * y_j \tag{2}$$

$$v_j = W_v * y_j \tag{3}$$

  Here, W_q, W_k, and W_v are learned weight matrices.

- Compute the Attention Weights: The attention weights are computed by applying a softmax function to the dot product of the query vector and the key vector for each input element. The attention weights $a_{i,j}$ are computed for each input element x_i and output element y_j as follows:

$$a_{i,j} = \text{sofmax}(\frac{q_i * k_j}{\sqrt{d_k}}) \tag{4}$$

  where d_k is the dimension of the key vectors

- Compute the Weighted Sum: The weighted sum of the input elements is computed by multiplying each input element by its corresponding attention weight and summing the results. The weighted sum c_j of the input elements is computed as follows:

$$c_j = \sum_i (a_{i,j} * v_i) \tag{5}$$

- Generate the Output: The weighted sum is used to generate the output element. The output element y_j is generated using the weighted sum c_j:

$$y_j = W_o * [c_j; y_{j-1}] \tag{6}$$

where,

$$[c_j; y_{j-1}]$$

is the concatenation of the weighted sum and the previous output element y_{j-1}, and W_o is a learned weight matrix.

Advantages of the Attention Mechanism over Other Methods: the attention mechanism has several advantages over other methods for processing sequential data, including: flexibility: The attention mechanism allows the network to selectively focus on different parts of the input when making predictions. This allows the network to adapt to different input sequences and capture complex relationships between different parts of the sequence; efficiency: The attention mechanism allows the network to process the input sequence in parallel, rather than sequentially. This can lead to significant speedups in training and inference times; interpretability: The attention weights provide a measure of the importance of each input element for a given output element. This can help to explain the network's predictions and provide insights into how it is processing the input sequence.

Self-attention, multi-head attetion and cross-attention There are several types of attention mechanisms that are commonly used in neural networks, including self-attention and multi-head attention. Here's a brief overview of each type:

- Self-Attention: Self-attention is an attention mechanism that computes the attention weights and the weighted sum using only the input sequence itself. In other words, it allows each element in the input sequence to attend to every other element in the sequence. Self-attention is particularly useful for tasks that require capturing long-range dependencies between different parts of the sequence. In the Transformer architecture, self-attention is used in the encoder and decoder layers to compute the contextualized representation of each input and output element.

- Multi-Head Attention: Multi-head attention is an extension of self-attention that allows the network to attend to different parts of the input sequence

in parallel. In multi-head attention, the input sequence is split into several smaller sequences, and self-attention is applied to each of these sequences separately. The outputs of each self-attention module are then concatenated and passed through a linear layer to generate the final output. The advantage of multi-head attention is that it allows the network to attend to different aspects of the input sequence simultaneously. This can lead to improved performance and faster convergence in training.

- Cross-Attention: Cross-attention is an attention mechanism that computes the attention weights and the weighted sum using two different input sequences. In the Transformer architecture, cross-attention is used in the decoder layers to compute the contextualized representation of each output element based on the input sequence. In cross-attention, the query vectors are computed from the output sequence, while the key and value vectors are computed from the input sequence. The attention weights are then computed based on the dot product of the query and key vectors, and the weighted sum is computed using the value vectors.

The different types of attention mechanisms offer different levels of flexibility and computational efficiency, depending on the specific requirements of the task at hand.

2.1.3 Encoder-Decoder Architecture

Definition The encoder-decoder architecture is a common neural network architecture used for sequence-to-sequence (seq2seq) tasks, such as machine translation, text summarization, and speech recognition. In the traditional encoder-decoder architecture, the input sequence is first fed into an encoder network, which generates a fixed-length vector representation of the input sequence. This vector representation, also known as the context vector, contains information about the input sequence that is relevant for generating the output sequence. The context vector is then fed into a decoder network, which generates the output sequence one element at a time. In the early days of seq2seq modeling, recurrent neural networks (RNNs) were commonly used as the encoder and decoder networks. In this case, the input sequence is fed into the encoder RNN, which generates a sequence of hidden states that encode the information about the input sequence. The final hidden state of the encoder RNN is then used as the context vector for the decoder RNN, which generates the output sequence one element at a time. However, RNNs suffer from the vanishing gradient problem and are slow to train, which limits their performance on long sequences. More recently, the Transformer architecture has emerged as a popular alternative to RNNs for seq2seq tasks. The Transformer uses self-attention to compute the context vector for the decoder, rather than relying on a fixed-length vector generated by an encoder. The input sequence is first passed through an encoder stack of self-attention and feedforward layers to generate a sequence of encoder outputs. The decoder stack then uses self-attention and cross-attention over the encoder outputs to generate the output sequence. The self-attention

mechanism allows the Transformer to capture long-range dependencies in the input sequence without suffering from the vanishing gradient problem, and the parallelizable nature of self-attention allows for faster training and inference. The encoder-decoder architecture is a powerful tool for seq2seq tasks, and the choice between RNN and Transformer architectures will depend on the specific requirements of the task at hand.

How it works The encoder-decoder architecture works by first encoding an input sequence into a fixed-length context vector using an encoder network, and then decoding this context vector into an output sequence using a decoder network. The encoder network takes as input a sequence of tokens (e.g., words, characters, or phonemes) and generates a sequence of hidden states that represent the input sequence. Each hidden state is generated by passing the current token and the previous hidden state through a non-linear activation function (e.g., a hyperbolic tangent or a rectified linear unit). The final hidden state of the encoder network is used as the context vector for the decoder network. The decoder network takes as input the context vector and generates the output sequence one token at a time. At each time step, the decoder network generates a hidden state based on the current input token and the previous hidden state, and then generates the output token based on this hidden state. This process is repeated until an end-of-sequence token is generated or a maximum output length is reached. In the traditional encoder-decoder architecture, the encoder and decoder networks are typically implemented as recurrent neural networks (RNNs), such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks. However, these networks can suffer from the vanishing gradient problem, which limits their performance on long sequences. To address this issue, the Transformer architecture was introduced. The Transformer uses self-attention instead of RNNs to encode and decode the sequences. Self-attention allows the model to capture long-range dependencies between the tokens in the sequence, while also being parallelizable and efficient. In the Transformer architecture, the input sequence is first passed through a stack of encoder layers, each of which applies self-attention and feedforward layers to the input sequence to generate a sequence of encoder outputs. The decoder network then uses self-attention and cross-attention over the encoder outputs to generate the output sequence. The self-attention mechanism allows the Transformer to capture long-range dependencies in the input sequence without suffering from the vanishing gradient problem, and the parallelizable nature of self-attention allows for faster training and inference. The encoder-decoder architecture is a powerful tool for sequence-to-sequence tasks, and the choice of encoder and decoder architecture will depend on the specific requirements of the task at hand.

How Transformer generates prediction The Transformer architecture uses the encoder-decoder architecture to generate predictions for a variety of sequence-to-sequence tasks, such as machine translation, text summarization, and question answering. In the Transformer, the input sequence is first passed

through a stack of encoder layers, each of which applies self-attention and feed-forward layers to the input sequence to generate a sequence of encoder outputs. These encoder outputs contain information about the input sequence that is relevant for generating the output sequence. The output sequence is then generated by the decoder network, which also consists of a stack of decoder layers. At each time step, the decoder takes the previously generated tokens (or the start-of-sequence token for the first time step) and the context vector generated by the encoder as input, and generates the next token in the output sequence. The context vector is generated by performing attention over the encoder outputs, weighted by the current state of the decoder. The attention mechanism used in the Transformer allows the model to focus on different parts of the input sequence at each time step, based on the current state of the decoder. This enables the model to capture long-range dependencies and generate accurate predictions for a variety of sequence-to-sequence tasks. During training, the model is typically trained to minimize a loss function that measures the difference between the predicted output sequence and the ground truth output sequence. The model parameters are updated using gradient descent, which iteratively adjusts the model parameters to minimize the loss function. The Transformer architecture uses the encoder-decoder architecture to generate predictions for sequence-to-sequence tasks, with the attention mechanism enabling the model to capture long-range dependencies and generate accurate predictions.

2.1.4 Transformer Components

Input Embedding The Input Embedding layer in the Transformer architecture is responsible for mapping the input tokens (e.g., words, characters, or phonemes) into continuous vector representations, also known as embeddings. These embeddings capture the semantic meaning of the tokens and allow the model to operate on a continuous vector space. In the Transformer, the input embeddings are first multiplied by a learned weight matrix, and then positionally encoded to capture the order of the tokens in the sequence. The position encoding is added to the embeddings and serves as a way for the model to distinguish between tokens based on their position in the sequence. The equation for the Input Embedding layer in the Transformer is:

$$E_{input} = Embedding(x) * \sqrt{d_{model}} + PE \tag{7}$$

where,

- x is the input token
- $Embedding$ is a function that maps the input token to its embedding vector
- $\sqrt{d_{model}}$ is a scaling factor used to prevent the embeddings from becoming too small or too large
- PE is the positional encoding vector for the input token

- E_{input} is the final input embedding vector for the input token

The dimensionality of the embedding vector is typically a hyperparameter that is set based on the size of the input vocabulary and the complexity of the task. In the Transformer architecture, the embedding dimensionality is often set to be the same as the hidden layer dimensionality, d_{model}, which is also a hyperparameter.

Positional Encoding Positional Encoding is used in the Transformer architecture to add information about the position of each token in the sequence to the input embeddings. This allows the model to distinguish between tokens based on their position in the sequence, even if the same token appears multiple times in different positions. The Positional Encoding function is defined as:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{8}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{9}$$

where,

- pos is the position of the token in the sequence
- i is the index of the dimension of the embedding vector
- d_{model} is the dimensionality of the embedding vector

The positional encoding is added to the input embeddings using element-wise addition. The choice of the positional encoding function is based on the intuition that low-frequency components represent global positional information, while high-frequency components represent local positional information. The use of sine and cosine functions with varying frequencies allows the model to capture different levels of positional information. The exact function and hyperparameters used for positional encoding can be tuned based on the task and dataset.

Encoder The Transformer Encoder component is responsible for encoding the input sequence into a sequence of hidden representations, which are then passed to the Transformer Decoder for generating the output sequence. The Transformer Encoder consists of a stack of N identical layers, each of which has two sub-layers: a Multi-Head Attention layer and a Positionwise Feedforward layer. The input sequence is first passed through an Input Embedding layer, which maps the tokens to their continuous vector representations. Then, the Positional Encoding is added to the input embeddings to capture the order of the tokens in the sequence. The Multi-Head Attention layer in the Encoder has three inputs: the Query matrix, the Key matrix, and the Value matrix. These matrices are generated from the input embeddings by linear transformations, and are used to calculate the attention scores between the input tokens. The output of the Multi-Head Attention layer is a weighted sum of the Value matrix,

where the weights are determined by the attention scores. The Multi-Head Attention layer can be expressed mathematically as:

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O \tag{10}$$

where,

- Q is the Query matrix
- K is the Key matrix
- V is the Value matrix
- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$ is the attention output for the i-th attention head
- W_i^Q, W_i^K, W_i^V are learnable weight matrices for the i-th attention head. These are hav been briefly introduced before in Eq. 1, Eq. 2 and Eq. 3
- W^O is a learnable weight matrix used to combine the outputs of the attention heads
- $Concat$ is a function that concatenates the outputs of the attention heads along the last dimension
- h is the number of attention heads
- $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$ is the attention mechanism between the Query, Key, and Value matrices

The Positionwise Feedforward layer in the Encoder applies two linear transformations with a ReLU activation function to each position independently and identically. The non-linear dependencies between the hidden states that the Positionwise Feedforward layer is designed to capture refer to the complex relationships between the embedded tokens in the sequence. Each token in the sequence is initially represented as an embedding vector, and these embeddings are then transformed through the multiple layers of the Transformer architecture to produce a final output sequence. The Positionwise Feedforward layer is an essential component of the Transformer architecture because it allows the model to capture non-linear relationships between the embedded tokens in the sequence. This is important because natural language is inherently complex, and the relationships between tokens in a sentence can be highly non-linear and difficult to model using linear techniques. By introducing non-linearity into the model through the use of the ReLU activation function, and by allowing the model to learn a non-linear mapping between the input and output sequences through the use of the two linear transformations, the Positionwise Feedforward layer enables the Transformer architecture to capture complex patterns in the data and achieve state-of-the-art performance on a wide range of natural language processing tasks.It can be expressed mathematically as:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{11}$$

where,

- x is the input vector
- W_1, b_1, W_2, b_2 are learnable weight matrices and biases

The output of each layer in the Encoder is passed as input to the next layer in the stack, allowing for the encoding of increasingly complex and abstract representations of the input sequence. The equations for the Encoder can be summarized as:

$$H_0 = E_{input} \tag{12}$$

$$H_i = MultiHead(H_{i-1}W_i^Q, H_{i-1}W_i^K, H_{i-1}W_i^V) + H_{i-1} \tag{13}$$

$$H_i = FFN(H_i) + H_i \tag{14}$$

where,

- E_{input} is the input sequence after the Input Embedding and Positional Encoding layers
- H_i is the output of the i-th layer in the Encoder
- W_i^Q, W_i^K, W_i^V and the learnable parameters of the Feedforward layers in the i-th layer
- The output of the Encoder is the final hidden representation sequence, H_N

Decoder The Transformer Decoder component is responsible for generating the output sequence, based on the encoded input sequence and the previously generated tokens. The Transformer Decoder also consists of a stack of N identical layers, each of which has three sub-layers: a Masked Multi-Head Attention layer, a Multi-Head Attention layer, and a Positionwise Feedforward layer. However, unlike the Encoder, the Decoder also has an additional Input Embedding layer for embedding the target sequence. The Masked Multi-Head Attention layer in the Decoder is similar to the Multi-Head Attention layer in the Encoder, but with a mask applied to prevent attending to future tokens. The Query, Key, and Value matrices in the Masked Multi-Head Attention layer are all generated from the previously generated tokens, up to the current time step. The Multi-Head Attention layer in the Decoder also attends to the Encoder output, in addition to the Masked Multi-Head Attention. The Query matrix in this case is generated from the previous layer in the Decoder, while the Key and Value matrices are generated from the Encoder output. This allows the Decoder to attend to relevant parts of the input sequence while generating the output sequence. The Positionwise Feedforward layer in the Decoder is the same as the one in the Encoder, see Eq. 11. The Input Embedding layer in the Encoder is responsible for embedding the tokens of the source sequence into a continuous vector space, where each token is represented as a dense vector. This layer

learns the embeddings of the source tokens during training. In contrast, the Decoder has to generate tokens in the target language, which are also represented as a sequence of embeddings. Therefore, the Decoder also has an additional Input Embedding layer that is responsible for embedding the tokens of the target sequence into a continuous vector space, where each token is represented as a dense vector. This layer also learns the embeddings of the target tokens during training. The embeddings of the target tokens generated by the Input Embedding layer are then fed into the subsequent layers of the Decoder, such as the Masked Multi-Head Attention layer, the Multi-Head Attention layer, and the Positionwise Feedforward layer, to generate the output sequence.

Output Layer The output layer is the final layer in the Transformer architecture and is responsible for producing the final output sequence. The output layer takes as input the final hidden state of the decoder, which has been processed through the multiple layers of the Transformer architecture, and produces a probability distribution over the vocabulary of the target language. The output layer typically consists of a linear transformation followed by a softmax activation function. The linear transformation projects the final hidden state of the decoder onto a high-dimensional space, and the softmax function normalizes the resulting vector to produce a probability distribution over the vocabulary. The output layer is designed to produce a probability distribution over the vocabulary of the target language, which allows the model to generate predictions for each token in the output sequence. During training, the model is typically trained to maximize the log-likelihood of the target sequence given the input sequence, which involves minimizing the cross-entropy loss between the predicted distribution and the true distribution over the target vocabulary. In summary, the output layer is an essential component of the Transformer architecture because it enables the model to generate predictions for each token in the output sequence by producing a probability distribution over the target vocabulary. The output layer is typically implemented using a linear transformation followed by a softmax activation function and is trained to maximize the log-likelihood of the target sequence given the input sequence.

2.1.5 Training and Inference

Backpropagation The Transformer is trained using backpropagation and gradient descent, which involves computing the gradients of the loss function with respect to the model parameters and updating the parameters in the direction of the negative gradient to minimize the loss.

The training process for the Transformer typically involves the following steps:

- Forward Pass: During the forward pass, the input sequence is fed into the encoder, and the output sequence is generated by the decoder. The output sequence is then compared to the target sequence using a loss function, such as cross-entropy loss.

- Backward Pass: During the backward pass, the gradients of the loss with respect to the model parameters are computed using the chain rule of calculus. The gradients are then backpropagated through the layers of the Transformer architecture, from the output layer to the input layer.
- Parameter Update: After computing the gradients, the model parameters are updated in the direction of the negative gradient using an optimization algorithm, such as stochastic gradient descent (SGD), Adam, or Adagrad. The learning rate hyperparameter determines the step size of the parameter update.
- Repeat: Steps 1-3 are repeated for multiple epochs, until the model converges to a satisfactory solution.

During training, the gradients are typically computed using a technique called teacher forcing, in which the decoder is fed the true target sequence as input at each time step, rather than the predicted sequence from the previous time step. Teacher forcing can speed up convergence during training, but can lead to suboptimal performance at inference time when the model is required to generate sequences without access to the true target sequence.

The loss function of the Transformer is typically the cross-entropy loss, which measures the dissimilarity between the predicted probability distribution over the target vocabulary and the true probability distribution over the target vocabulary. The cross-entropy loss is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}) \tag{15}$$

where N is the number of training examples, M is the size of the target vocabulary, y_{ij} is the true probability of the j-th token in the i-th target sequence, and p_{ij} is the predicted probability of the j-th token in the i-th target sequence. During training, the goal is to minimize the cross-entropy loss with respect to the model parameters, which can be achieved using backpropagation and gradient descent. The gradients of the loss with respect to the model parameters can be computed using the chain rule of calculus and are backpropagated through the layers of the Transformer architecture to update the model parameters. The cross-entropy loss is a commonly used loss function in natural language processing (NLP) tasks, such as machine translation, text classification, and language modeling. It is well-suited for these tasks because it provides a measure of the dissimilarity between the predicted and true probability distributions over the target vocabulary, which is the main objective of NLP tasks.

The update equation for a parameter θ at iteration t is:

$$\theta_{t+1} = \theta_t - \eta \cdot g_t \tag{16}$$

where η is the learning rate, and g_t is the gradient of the loss with respect to θ at iteration t. The learning rate determines how quickly the parameters

are updated in response to the gradients. A high learning rate can result in large updates that overshoot the optimal parameter values, while a low learning rate can result in slow convergence and getting stuck in local minima. By iteratively repeating these steps on a large dataset of input-target pairs, the model gradually learns to produce accurate translations.

Inference Inference refers to the process of using a trained Transformer model to generate predictions on new, unseen data. Here is a brief overview of how the Transformer generates predictions during inference:

- Encoder: We first feed the input sequence into the Transformer encoder to obtain the encoded representation of the input sequence.
- Start token: We start with a special token (e.g., <s> or [START]) as the first token in the target sequence.
- Decoder: We feed the encoded representation and the current target sequence into the Transformer decoder to obtain the predicted probability distribution over the target vocabulary for the next token in the sequence.
- Sampling: We sample a token from the predicted probability distribution using a sampling strategy such as greedy search, beam search or top-K sampling. This sampled token becomes the next token in the sequence.
- Repeat: We repeat steps 3 and 4 until we reach a special end-of-sequence token (e.g., </s> or [END]) or a predefined maximum length.

During inference, the Transformer uses the same encoder and decoder architecture as during training, but with a few key differences. First, during inference, we do not use teacher forcing, which means we do not feed the ground truth target sequence as input to the decoder at each time step. Instead, we use the previously generated token as the input to the decoder at each time step. This means that errors can accumulate as the model generates longer sequences. Second, during inference, we do not update the parameters of the model using gradient descent. Instead, we use the fixed parameters learned during training to generate predictions. Finally, during inference, we use techniques such as beam search or sampling to generate multiple possible predictions, which we can then rank and select the best one based on some evaluation metric such as BLEU score.

2.2 Sparse Retrieval

A sparse representation is a high-dimensional vector where most of the values are zero or very small. This is often the case with traditional weighting schemes like TF-IDF [14] or BM25 [15] or BM25+ [16], which consider only the presence or absence of individual terms in a document or query. The resulting vectors are sparse because only a small subset of the available terms will be present in each document or query, and the values for those terms will be non-zero or relatively

high. Sparse representations are generally more efficient to store and process than dense representations, especially when the number of dimensions is very large, as is often the case with text data. However, they may not capture the full complexity of the data and may not perform as well as dense representations on certain tasks, such as natural language understanding and text generation.

Sparse retrieval methods are mainly based on relevant, similarity between two documents, or between a query and documents based on keywords that appear in both. Sparse Retrieval is famous for its classic algorithms including algorithms like BM25 , TFIDF , of which is the best algorithm in this approach. The biggest disadvantage of Sparse Retrieval is that the representation of the vocabulary will directly and significantly affect the performance of the algorithms, namely lexical mismatching. For example, “Civil law” and “civil law” have the same meaning but are two different strings because the "c" character is capital. In addition, to use these algorithms, the input string must be processed by removing stopwords, which means that the grammatical structure of the sentence will not be considered, a very important part of the semantic representation. of the sentence. Especially, for Vietnamese, compound words in Vietnamese show different meanings if we separate two compound words into separate words. Depending on the language, the sparse retrieval approach must have a specific treatment in each language.

TF-IDF TF-IDF stands for "Term Frequency-Inverse Document Frequency" and is a commonly used technique in natural language processing (NLP) for text data preprocessing. It is a statistical method that is used to evaluate the importance of a term within a document or corpus.

TF-IDF considers two factors for each term in a document or corpus:

- Term Frequency (TF): This measures the frequency of a term within a document or corpus. It is calculated as the number of times a term appears in a document divided by the total number of terms in the document.
- Inverse Document Frequency (IDF): This measures the rarity of a term across the entire corpus. It is calculated as the logarithm of the total number of documents in the corpus divided by the number of documents containing the term.

The formula for calculating TF-IDF is as follows:

$$\text{TF-IDF}(t, d, N) = \text{TF}(t, d) * \text{IDF}(t, N) \tag{17}$$

where

$$TF(t, d) = \frac{\text{number of times term } t \text{ appears in document } d}{\text{total number of terms in document } d} \tag{18}$$

$$IDF(t, N) = \ln(\frac{N}{n_t}) \tag{19}$$

where, N is the total number of documents in the corpus, and n_t is the number of documents that contain the term t.

TF-IDF can be used to encode each document in a corpus into a vector representation, often referred to as a document embedding. The TF-IDF scores for each term in a document can be combined to form a vector that represents the document in a high-dimensional space. This vector can then be used as input for various machine learning models or for similarity calculations between documents.

To create a document embedding using TF-IDF, we first need to build a vocabulary of all the unique terms in the corpus. Then, we can calculate the TF-IDF score for each term in each document, as shown in the example I provided earlier. Finally, we can represent each document as a vector where each element corresponds to the TF-IDF score of a term in the vocabulary.

BM25 BM25 (Best Match 25) is a ranking function used in information retrieval that is a variant of the TF-IDF weighting scheme. It is often used in search engines to rank the relevance of documents to a particular search query. Like TF-IDF, BM25 calculates a score for each document in a corpus based on the frequency of its query terms in the document, but it also takes into account document length and term frequency saturation. The formula for BM25 score is given by:

$$\text{BM25}(q,d) = \sum_{i=1}^{|q|} \text{IDF}(q_i) \cdot \frac{f(q_i,d) \cdot (k+1)}{f(q_i,d) + k \cdot (1 - b + b \cdot \frac{|d|}{\text{avgdl}})} \tag{20}$$

where

- q is the query
- d is a document in the corpus
- $|q|$ is the length of the query
- $\text{IDF}(q_i)$ is the IDF of the i-th query term q_i
- $f(q_i, d)$ is the frequency of the i-th query term q_i in document d
- k and b are tunable parameters
- $|d|$ is the length of the document d
- avgdl is the average document length in the corpus

The BM25 formula differs from TF-IDF in several ways. Firstly, it uses a non-linear function to calculate term frequency weights, where the weight increases logarithmically with the frequency of the term. Secondly, it incorporates document length normalization through the term $(1 - b + b\frac{|d|}{avgdl})$, which decreases the weight of terms that occur frequently in long documents. Finally, the parameters k and b can be tuned to adjust the importance of term frequency and

document length normalization in the calculation. Overall, BM25 is a more advanced weighting scheme than simple TF-IDF and has been shown to produce better results in many information retrieval tasks.

Document length can affect the score in information retrieval because longer documents tend to have more occurrences of any given term simply due to their greater length. This can lead to longer documents being ranked higher than shorter documents even if they contain less relevant information for a given query. To address this issue, the BM25 formula incorporates document length normalization through the term $(1-b+b\frac{|d|}{avgdl})$. This term decreases the weight of terms that occur frequently in long documents, and thus helps to mitigate the impact of document length on the score. By doing so, it allows shorter documents to compete more fairly with longer documents in the ranking.

The value of the b parameter in the BM25 formula controls the strength of the document length normalization. When b is set to 0, there is no length normalization and document length has no effect on the score. When b is set to 1, the length normalization is fully applied and longer documents are penalized relative to shorter documents. Intermediate values of b provide varying degrees of length normalization. Overall, document length normalization is an important aspect of information retrieval ranking algorithms like BM25, as it helps to ensure that documents are ranked based on their true relevance to a given query, rather than simply their length.

BM25+ BM25+ (also known as BM25F) is an extension of the BM25 algorithm that takes into account document fields, which are different sections of a document that may have different importance for a given query. In BM25+, each term in the query is assigned a weight based on its frequency in the query and the inverse document frequency (IDF) of the term in the corpus. The weights are then combined to produce a score for each document. However, in addition to the term weights, BM25+ also assigns weights to document fields based on their importance for the query. Here's a high-level overview of how BM25+ works:

- Document fields: Each document is divided into fields, which can be things like the title, author, or body of the document.
- Field weights: A weight is assigned to each field based on its importance for the query. For example, the title field might be given a higher weight than the body field if the query is expected to match more often in the title.
- Query processing: The query is processed as in the basic BM25 algorithm, with weights assigned to each term based on its frequency in the query and the IDF of the term in the corpus.
- Scoring: The weights for each term and field are combined to produce a score for each document. The score is a weighted sum of the term weights,

where the weight for each term is multiplied by the weight of the field it appears in.

- Ranking: The documents are ranked in descending order based on their scores, and the top-ranked documents are returned as the search results.

BM25+ can be a more effective ranking algorithm than basic BM25 when documents contain multiple fields, since it takes into account the varying importance of each field for the query. However, BM25+ also has more tunable parameters than basic BM25, since weights must be assigned to each field. BM25+ combines the basic BM25 formula with field weights. Here's the formula for computing the BM25+ score for a document:

$$\text{BM25+}(q,d) = \sum_{i=1}^{|q|} \text{IDF}(q_i) \cdot \frac{w_i \cdot f(q_i,d) \cdot (k+1)}{f(q_i,d) + k \cdot (1 - b + b \cdot \frac{|d|}{\text{avgdl}})} \tag{21}$$

where

- q is the query
- d is a document in the corpus
- $|q|$ is the length of the query
- $\text{IDF}(q_i)$ is the IDF of the i-th query term q_i
- $f(q_i, d)$ is the frequency of the i-th query term q_i in document d
- k and b are tunable parameters
- $|d|$ is the length of the document d
- avgdl is the average document length in the corpus
- w_i is the weight assigned to the i-th query term q_i

This equation represents the BM25+ ranking function, which is an extension of the BM25 function that allows for query term weighting. The w_i term represents the weight assigned to each query term, which can be used to adjust the importance of different terms in the query. Using a pre-trained deep neural language model, it is possible to encode questions or supporting documents into a continuous latent semantic embedding vector,allowing for more accurate similarity matching and ranking of relevant passages. Dense passage retrieval is helpful because it allows for the efficient retrieval of relevant passages or documents in response to a given query. However, dense passage retrieval uses deep neural language models to encode queries and documents into continuous latent semantic vectors, allowing for more accurate similarity matching and ranking of relevant passages. The success in recent researches such as [17] [18] or [19] is contributed by dense retrieval approaches.

2.3 Dense Retrieval

Using a pre-trained deep neural language model, it is possible to encode questions or supporting documents into a continuous latent semantic embedding vector,allowing for more accurate similarity matching and ranking of relevant passages. Dense passage retrieval is helpful because it allows for the efficient retrieval of relevant passages or documents in response to a given query. However, dense passage retrieval uses deep neural language models to encode queries and documents into continuous latent semantic vectors, allowing for more accurate similarity matching and ranking of relevant passages. The success in recent researches such as [17] [18] or [19] is contributed by dense retrieval approaches.

2.4 Cross-encoder approaches

The cross-encoder consists of a backbone language model, which can be BERT [20], RoBERTa [7], or any other transformer encoder model. It is able to capture global interactions between a query and a document [21][22]. The input consists of a pair of a query and a document, which are passed through the backbone language model to generate a joint representation that captures the relationship between the two inputs. However, the cross-encoder approach also has some drawbacks, such as higher computational complexity and longer training times compared to the dual-encoder approach. Nevertheless, its advantages in capturing global interactions between inputs make it a powerful approach for dense retrieval in information retrieval and related fields.

2.5 Dual-encoder approaches

The dual-encoder approach involves two backbone language models, typically also transformer encoder models. One model is trained to encode queries, while the other is trained to encode documents. The dual-encoder approach maps input queries and output targets to a shared vector space, where the inner products of the query and target vectors can be used as a reliable similarity function. In practice, dual encoders accomplish the ability to scale to a large number of targets through two mechanisms: sharing weights among targets by means of a parametric encoder, and employing a scoring function based on inner products that is efficient. Therefore, it is a potential research topic of [23],[24]

2.6 Sequence-to-Sequence (Seq2Seq) for question answering

In an extractive question answering task, the goal is to identify the answer span within a given context that best answers a question. While encoder models are currently the preferred choice for extractive question answering, sequence-to-sequence (Seq2Seq) is also proved its great performance in extractive question answering via [11] [25] [26]. Recently, a novel method for applying Transformer models to extractive question answering tasks has been proposed [27]

2.7 Beam search

Beam search is a popular decoding algorithm used in natural language processing for generating text. It works by iteratively generating the most probable next token based on a language model, while keeping track of a fixed number (k) of most probable sequences (i.e., "beams") at each time step. Here is the mathematical equation that describes how Beam search works:

At each time step t:

- Initialize the set of k most probable sequences: $H_t^{(0)} = \emptyset$
- For each sequence $h_{t-1}^{(i)} \in H_{t-1}^{(i)}$, generate the top k most probable next tokens and append them to the sequence to form k new candidate sequences: $H_t^{(i)} = h_{t-1}^{(i)} + y : y \in top_k(P(y|h_{t-1}^{(i)}))$
- Sort the k * k candidate sequences by their probability based on the language model score and keep only the top k: $H_t^{(i)} = top_k(H_t^{(i)})$
- Repeat steps 2-3 for each sequence in $H_{t-1}^{(i)}$ to obtain $H_t^{(i+1)}$
- Terminate the search if the end-of-sequence token is generated or the maximum length is reached. Otherwise, go to step 2 for the next time step.
- Return the most probable sequence among all sequences in H_t at the final time step.

In the above equation, $H_t^{(i)}$ represents the set of the i-th most probable sequences at time step t, $P(y|h_{t-1}^{(i)})$ represents the conditional probability of token y given the previous sequence $h_{t-1}^{(i)}$, and $top_k(S)$ represents the top k elements in set S based on a score or ranking function.

2.8 Contrastive learning in Information Retrieval

2.8.1 Query-document matching

Information retrieval is the task of retrieving relevant documents from a large corpus in response to a user query. The goal of information retrieval is to match the user's information need as expressed in the query with the relevant documents in the corpus.

In an information retrieval system, the user submits a query, which is typically a short piece of text describing their information need. The system then searches a large corpus of documents, such as a collection of web pages, news articles, or scientific papers, to find the documents that are most relevant to the query.

The relevance of a document to a query is typically assessed using some relevance criteria, such as keyword overlap, term frequency, or other statistical measures. The documents that are most relevant to the query are then returned to the user as the search results.

Information retrieval has a wide range of applications, including web search, document search, patent search, legal search, and many others. The goal of information retrieval is to help users find the information they need quickly and efficiently, by matching their queries with the most relevant documents in the corpus.

2.8.2 Learn a good representation for queries and documents

One way to improve information retrieval is to learn a good representation of documents and queries that captures their similarity or relevance. The idea is to transform the raw text of documents and queries into a numerical representation that can be used by machine learning algorithms to model the similarity or relevance between them.

The representation should ideally capture the underlying semantics and meaning of the text, rather than just its surface features. For example, two documents that are about the same topic but use different words or phrases to describe it should be represented as similar in the representation space.

Learning a good representation of documents and queries can be done using various techniques, such as:

- Vector space models: This approach represents documents and queries as vectors in a high-dimensional space, where each dimension corresponds to a different term or word. The similarity between two documents or queries is then computed using various distance measures, such as cosine similarity or Euclidean distance.
- Topic models: This approach represents documents as mixtures of latent topics, where each topic is a probability distribution over the words in the vocabulary. The similarity between two documents is then computed based on their overlap in topic distributions.
- Neural network-based models: This approach uses neural networks to learn a non-linear mapping from the raw text of documents and queries to a continuous vector space. The network is typically trained using a supervised or unsupervised learning objective, such as maximum likelihood or contrastive learning.

By learning a good representation of documents and queries, we can improve the performance of information retrieval systems by better capturing the semantic and contextual information in the text, and by enabling more sophisticated ranking and retrieval algorithms that take into account the similarity or relevance between the query and the documents in the corpus.

2.8.3 Contrastive learning

Contrastive learning is a popular unsupervised learning (most of cases) approach for learning representations because it can learn useful representations without

the need for explicit supervision or labeled data. Contrastive learning is a self-supervised learning method that learns representations by contrasting similar and dissimilar pairs of data.

In contrastive learning, we start by creating pairs of similar and dissimilar data points. For example, we can create a pair of similar data points by taking two different augmentations of the same image, or we can create a pair of dissimilar data points by taking an image and a completely different image.

The idea is to learn a representation that maps similar data points close together in the embedding space and dissimilar data points far apart. This is achieved by optimizing a contrastive loss function that encourages the model to minimize the distance between similar data points while maximizing the distance between dissimilar data points.

Contrastive learning has shown to be effective in a variety of tasks, including image classification, object detection, and natural language processing. It has also been shown to be effective in learning representations for information retrieval tasks, such as dense passage retrieval.

One reason why contrastive learning is popular is because it does not require labeled data, which can be expensive and time-consuming to obtain. Instead, it can use unlabeled data, which is typically abundant and easy to obtain. Additionally, contrastive learning can leverage the vast amounts of unlabeled data available on the internet to learn general-purpose representations that can be fine-tuned on specific tasks. This makes contrastive learning an attractive approach for learning representations for a wide range of applications, including information retrieval.

2.8.4 Distinguish between positive and negative pairs of text

Contrastive learning works by learning a representation that maps similar data points close together in the embedding space and dissimilar data points far apart. This is achieved by optimizing a contrastive loss function that encourages the model to minimize the distance between similar data points while maximizing the distance between dissimilar data points. In the context of learning representations for information retrieval, the data points can be queries and documents, and the goal is to learn a representation that maps relevant queries and documents close together and irrelevant queries and documents far apart.

Formally, let's assume we have a set of queries Q and a set of documents D, and we want to learn a representation function $f(.)$ that maps queries and documents to a common embedding space, i.e., $f(Q)$ and $f(D)$ are embeddings of the queries and documents, respectively. We start by creating pairs of positive and negative examples, where a positive pair consists of a query and a document that are relevant, and a negative pair consists of a query and a document that are irrelevant. For each positive pair (q,d), we create a negative pair (q,d'), where d' is a document that is not relevant to the query q. The contrastive loss function is defined in Eq. 36.

The metric which is used in the loss function, similarity score is usually cosin similarity. However, it is possible to replace cosine similarity with other

similarity scores in the contrastive loss function. The choice of similarity score depends on the nature of the data and the task at hand. Cosine similarity is a commonly used similarity score in contrastive learning because it is a robust and efficient way to measure the similarity between two vectors in high-dimensional spaces. It is particularly effective when the data is sparse, and the magnitude of the vectors is not important. However, depending on the task, other similarity scores can be used. For example, in image retrieval, the Euclidean distance or the Manhattan distance can be used instead of cosine similarity. In text retrieval, the Jaccard similarity or the Dice similarity can be used instead of cosine similarity. The choice of similarity score can be problem-specific and may require experimentation to find the best option. In general, the similarity score should be chosen based on its ability to capture the semantic similarity between the data points and its compatibility with the chosen representation function.

The loss function encourages the model to minimize the distance between the embeddings of the query and relevant document while maximizing the distance between the embeddings of the query and irrelevant documents. In other words, it tries to pull the embeddings of positive pairs closer together while pushing the embeddings of negative pairs farther apart. During training, the model learns to distinguish between positive and negative pairs by optimizing the contrastive loss function using stochastic gradient descent or other optimization algorithms. The learned representation can then be used for dense passage retrieval by computing the similarity between the query and all documents in the corpus using the learned embeddings.

Based on contrastive loss function in Eq. 36. q_{pos} represents a positive query, d_{pos} represents a positive document that is relevant to the query, and D_{neg} represents a set of negative documents that are irrelevant to the query. The function f is a scoring function that takes as input a query-document pair and outputs a relevance score. The loss function is computed as the negative logarithm of the fraction $\frac{e^{f(q_{pos},d_{pos})}}{e^{f(q_{pos},d_{pos})}+\sum_{d_{neg}\in D_{neg}} e^{f(q_{pos},d_{neg})}}$. The numerator $e^{f(q_{pos},d_{pos})}$ is the exponential of the relevance score between the positive query and the positive document, while the denominator $\sum_{d_{neg}\in D_{neg}} e^{f(q_{pos},d_{neg})}$ is the sum of the exponentials of the relevance scores between the positive query and all negative documents. The contrastive loss function penalizes negative samples by making the denominator in the fraction large. The denominator sums over the similarity scores between the query and all negative documents, which are chosen to be dissimilar to the query. When the similarity score between the query and a negative document is low, the corresponding term in the sum becomes a small positive number. However, when summed over all negative documents, these terms can add up to a large number, making the denominator in the fraction large. As a result, the value of the fraction approaches 0, and the log of the fraction approaches a large negative number. This large negative value is then multiplied by -1 to give a positive loss for the negative pair. By minimizing this positive loss, the model is encouraged to learn representations that push dissimilar documents or queries apart in the representation space.

2.8.5 Positive and negative pairs of text sampling

To sample positive and negative pairs as training data for contrastive learning in information retrieval, we need to have a set of queries and their relevant documents. One common approach is to use labeled data, where the relevance of each document to a query is provided as a label. Here is a general approach to sample positive and negative pairs:

- For each query, create a set of documents that are labeled as relevant to the query.
- Create a set of negative documents for each query by sampling a fixed number of documents from the corpus that are not labeled as relevant to the query.
- For each query and its relevant document, create a positive pair (question, relevant document).
- For each query and negative document, create a negative pair (question, irrelevant document).
- Repeat steps 3 and 4 for all queries q and their relevant and negative documents.

The number of negative documents sampled for each query can be tuned depending on the dataset size and the difficulty of the task. In general, a larger number of negative documents can make the training more challenging but also more effective. It is important to note that this approach assumes the availability of labeled data, which is not always the case in information retrieval. In such cases, unsupervised methods such as clustering or density-based sampling can be used to generate positive and negative pairs based on the similarity of the queries and documents in the corpus.

3 PROJECT MANAGEMENT PLAN

3.1 Overview

Looking at Figure , we can see the work that our team has done in the past 14 weeks:

- In the first few weeks, the main tasks were searching for literature and studying methods from related articles to gain different perspectives and details.
- In weeks 2 and 3, the team's main task was collecting appropriate datasets for the topic "Vietnamese legal document retrieval" specifically Zalo Legal 2021 dataset and also collecting data from reputable legal websites. The data was then processed by filtering noise and rearranging to fit the next training phase.
- From week 3 to 7, the team focused on analyzing query architectures and evaluating their effectiveness such as DPR or Condenser. Additionally, three experiments from versions V.0.1, V.0.2, and V.1.0 were completed to gain a comprehensive overview of the chosen model which was Condenser due to feasibility and efficiency on the Vietnamese language.
- From week 8 to 11, our work was to find hardware solutions such as renting and setting up GPUs to ensure progress in our training. We also created a small demo product for our project. Finally, we developed a "Question Answering" system to increase efficiency and provide better visualization of the results from previous models.
- In the remaining weeks, we evaluated the results and summarized our report, as well as wrote a paper for the scientific conference in Indonesia, ISICO 2023.

3.2 Work Details

Look at Figure . Based on the plans outlined for all weeks, we will assign tasks equally to both members. The agreed-upon tasks will be consistently implemented to ensure that both members can complete their work well while also understanding how the other person's work operates.

- In the early weeks, research work will be conducted in parallel by both members, and they can agree on the optimal method and approach for developing or training the model.
- Regarding data processing, Nhat will be primarily responsible for ensuring that the data is suitable and optimized for model training. However, data research and exploration from websites will be carried out by both members, and agreements will be made on how to handle the data.

Table 1: Over about timeline

Timeline	
Week 1	0. Literature review, including approaches to the topic, relevant articles, and data for the project.
Week 2	1.1 Data processing: Find relevant data sources and collect data from legal websites.
Week 3	1.2 Data processing: Standardize the data for training the models.
Week 4	2.1 Research on models and architectures for "Vietnamese legal text retrieval" such as DPR, Condenser.
Week 5	2.2 Feasibility study of different approaches and selection of the most suitable and effective model for the "Legal Text Retrieval" step.
Week 6	2.3 Model training: Create the test table V.0.1.
Week 7	2.4 Create a web demo..
Week 8	3. Find ways to improve the model and re-plan for the remaining part of the project.
Week 9	4.1 Model training and creation of the V.0.2 improved version.
Week 10	4.2 Rent a GPU server and install the server.
Week 11	4.3 Model training and creation of the V.1.0 improved version. Research and train for QA model
Week 12	5.1 Write the project report, summarize the best results to write the report for the scientific conference..
Week 13	5.2 Refine the report and create slides for the final project defense.
Week 14	5.3 Prepare the project report.

- Regarding the query model, Khang will be the main responsible person. However, we will still study two query structures, DPR and Condenser.

Table 2: Members's work details

STT	**Nguyen Hoang Gia Khang (SE150829)**	**Nguyen Minh Nhat (SE150958)**
Week 1	Research literature, including approaches to the topic, relevant articles, and data for the project	Research literature including approaches, papers on the topic, and data for the topic
Week 2	Collect data from "vbpl.vn"	Collect data from "lawnet.vn"
Week 3	Aggregate and process collected data	Preprocess data for training models
Week 4	Research Condenser	Research DPR
Week 5	Run Condenser tests for legal queries in Vietnamese (feasible)	Run DPR experiment for legal query in Vietnamese (not feasible)
Week 6	Create a V.0.1 version	Research Condenser
Week 7	Create a web page to run the product	Prepare metrics for evaluating the model
Week 8-9	Research and evaluate methods to improve the model	Train and create the V.0.2 improvement version
Week 10-11	Research and train for QA model	Train and create the V.1.0 improvement version
Week 12-13-14	Prepare the presentation for the conference	Prepare the report

Khang will develop the first version, V.0.1, and hand it over to Nhat for further development in subsequent versions, V.0.2 and V.1.0.

- For the remaining parts, we will focus on researching and training the "Question Answering" model, writing a thesis report, writing articles for scientific conferences, and creating a prototype to introduce the product.

Data source	# articles	# sentences
vbpl.vn	157323	5213346
lawnet.vn	131903	4182150

Figure 3: Crawled data information.

4 MATERIALS AND METHODS

4.1 Materials

4.1.1 Dataset

Overview With regard to " Answering Legal Questions by Learning Neural Attentive Text Representation" papper [5] which they built two datasets: 1) the legal document corpus, which contains Vietnamese legal documents; and 2) the QA dataset, which contains a set of legal questions (queries) and a list of relevant articles for each question. The raw legal documents were first crawled from the official online sites "https://vbpl.vn", "https://thuvienphapluat.vn/". The queries were collected from the legal advice websites as "https://hdpl.moj.gov.vn" and "https://hethongphapluat.com/hoi-dap-phap-luat.html". At first, both the title and the content are included in each query. They analyzed and discovered that the content is frequently lengthy and ambiguous. As a result, they filtered out content parts, rewrote informative titles, and kept only the good titles in the dataset. The entire collection of Vietnamese legal documents, which includes multiple versions of each law and regulation, is the raw data for the legal document corpus. they sifted through the repetitive old renditions and just held and planned the solutions to the as of now viable articles. With the assistance of lawyers, this procedure was carried out. In the end, they got the legal document corpus, which had 8,586 documents and 117,545 articles, and the query dataset, which had 5,922 queries and the articles that were relevant to them. The query dataset's statistics are presented. Each query has 1.6 relevant articles and averages 12.5 words (or 17.3 syllables).

Besides, The task of retrieval-based legal question answering at the article level, which presents several challenges in comparison to the task at the document level, is the focus of this paper. The number of legal documents is significantly less than the number of articles. In addition, distinguishing articles within the same document is extremely challenging due to the common vocabulary and common focus. An example legal question and the expected response, Article 651 from the 2015 Code of Civil Law of Vietnam, are shown in the figure 5. They discovered that only a few sentences in an answer article contain relevant information and that only a few phrases in such sentences are matched to the question by investigating legal questions and articles. They present a neural attentive text representation technique based on attention mechanisms

Sentence	Perplexity
Chây ì nộp phạt nguội. (Slow payment of fines)	277.18
Nếu chồng chị muốn theo em thì chị cho đi luôn. (If my husband wants to follow you, I'll let him go.)	3383.04
Trong khi Tư khai không dùng số điện thoại này. (While Tu say that he did not use this phone number.)	404.96
Nếu bị cáo trốn thì HĐXX tạm đình chỉ vụ án và yêu cầu CQĐT truy nã bị cáo. (If the defendant escapes, the trial panel shall temporarily suspend the case and request the investigating agency to pursue the accused.)	35.01
Bên cạnh đó, Thông tư 64/2017 cũng sửa đổi, bổ sung nội dung liên quan đến giấy đăng ký xe, biển số xe. (In addition, Circular 64/2017 also amends and supplements contents related to vehicle registration papers and license plates.)	87.62
Thị trấn biển Italia giao bán nhà với giá 1 USD. (The Italian beach town sells houses for $1.)	1322.54
Hai bên đã ký hợp đồng công chứng. (The two parties have signed a notarized contract.)	99.30
Ông Kính, bà Liễu, bà Thơm có yêu cầu bồi thường thiệt hại. (Mr. Kinh, Ms. Lieu and Ms. Thom have claims for damages.)	152.27

Figure 4: Example of in-domain data selection.

Question	Do stepchildren have rights of inheritance from the deceased father where there is no will?
Answer	Article 651 from the Code of Civil law of Vietnam (2015).
Article content	Article 651. Heirs at law 1. Heirs at law are categorized in the following order of priority: a) The first level of heirs comprises: spouses, biological parents, adoptive parents, offspring and adopted children of the deceased; b) The second level of heirs comprises: grandparents and siblings of the deceased; and biological grandchildren of the deceased; c) The third level of heirs comprises: biological great-grandparents of the deceased, biological uncles and aunts of the deceased and biological nephews and nieces of the deceased. 2. Heirs at the same level shall be entitled to equal shares of the estate. 3. Heirs at a lower level shall be entitled to inherit where there are no heirs at a higher level because such heirs have died, or because they are not entitled to inherit, have been deprived of the right to inherit or have disclaimed the right to inherit.

Figure 5: A sample in the dataset with highlighted parts in [5]

and convolutional neural networks to capture these properties. While the first component is meant to extract important information from the input question and legal documents, the second component is used to match and present the most important parts.

For the paper " Miko Team: Deep Learning Approach for Legal Question Answering in ALQAC 2022" [3], They finetuned RoBERTa with 4GB of legal text data to enhance its performance in the legal domain. They use two distinct approaches to gather this data: Collected directly from 2 websites and Extract sentences close to the legal topic from the news corpus. Firstly, Table I provides the collected data for the first method, which includes the number of articles (articles) and sentences (sentences). This number represents the number of sentences that are retained following the selection process (for example. removing redundant and non-Vietnamese sentences). Secondly, In order to extract legal documents from the news corpus using the second approach, which is based on the work in [23], we first create a collection of legal documents known as "in-domain" data. On this in-domain dataset, we then construct a statistical language model (base language model). Select sentences in another corpus whose perplexity score falls within the threshold by utilizing this model to evaluate the score for each sentence. A popular metric for evaluating language models is perplexity. We can determine how effective our language model is in our data domain using the perplexity score. Since it was learned from "in-domain" data, we assume that our base language model is adequate in this instance.

Specific works In terms of our application, Data on law in general or Vietnamese law in particular is still quite limited and rarely publicly available. Regarding the details of datasets for Vietnamese law, currently only Zalo Corporation's dataset from the Zalo Legal 2021 competition is widely available. In recent competitions on this topic, high-ranking teams often use this dataset to finetune their language models such as VinAI's Phobert, FPT's viBert, or the multilingual RoberTa model. However, when examining the highest-ranking teams, we see that in addition to using the Zalo Legal dataset, they often crawl data from reputable Vietnamese law websites such as "vbpl.vn" or "lawnet.vn" which can reach up to 4GB of data in papper [3] and are preprocessed before being used for model training.

However, with limited resources, we stopped collecting data when we reached 3GB of data from those websites. In addition to the main task of "Legal Query in Vietnamese", we also developed another extension called "Question and Answer". As shown in some recent articles such as [4] and [5], they also collected "Question and Answer" data from corresponding legal websites, but these data involved participation and supervision from experienced lawyers or legal experts who could evaluate in detail and remove unnecessary parts in the process of preprocessing data for subsequent model training stages. Within our capacity, we could also collect up to 12GB of data, but we do not have the resources to train and have legal experts evaluate the quality of the collected data sets.

4.1.2 Framework and Libraries

Pytorch Originally created by Meta AI and now a part of the Linux Foundation, PyTorch is a machine learning framework built on the Torch library and

used for applications like computer vision and natural language processing. It is open-source software that is available for free under a modified BSD license. PyTorch features a C++ interface, even though the Python interface is more refined and the main focus of development. Using PyTorch, a programmer can easily create a sophisticated neural network because its primary data format, Tensor, is a multi-dimensional array similar to Numpy arrays. Due to PyTorch's flexibility, speed, and ease of use, it is becoming more and more popular in both the business world and among researchers. PyTorch is one of the best deep learning tools. For a variety of reasons, PyTorch has emerged as one of the top machine learning frameworks: Widely Available; accelerated model development; Quick training periods; Support for High-Quality GPU Training and robust ecosystem

Transformers Thousands of pretrained models are available in Transformers to carry out tasks on several modalities, including text, vision, and audio. Text, for tasks like text categorization, information extraction, question answering, summarization, translation, and text synthesis in more than 100 languages are all tasks that may be applied to these models; Audio for activities like speech recognition and audio classification; images for activities like picture classification, object detection, and segmentation. With the help of Transformers' APIs, you can utilize those pretrained models on a given text rapidly, hone them using your own datasets, and then distribute the results to the community. In addition, each Python module specifying an architecture is completely independent and can be changed to allow for quick research experiments. The three most well-liked deep learning libraries, Jax, PyTorch, and TensorFlow, are supported by Transformers and seamlessly integrate with one another. It is simple to build your models with one and then load them with the other for inference.

Python Vietnamese Toolkit (pyvi)[28] A quick and accurate NLP annotation pipeline for Vietnamese, VnCoreNLP uses named entity recognition (NER), dependency parsing, word segmentation, and POS tagging to provide rich linguistic annotations. It's not necessary for users to install external dependencies. Both the command line and the API can be used by users to launch processing pipelines. The following articles are relevant and detail the general design and experimental outcomes of VnCoreNLP: [28], [29], [30].

4.1.3 Hardware

Cloud Service Free GPU-enabled platforms include Google Colab and Kaggle. However, there are still many limitations affecting the work progress. Specifically, Google Colab with the regular version only allows the GPU to be used in a certain amount of computing, for the Retriever module, it can only be trained for a period of 4-5 hours. For the colab pro version, the gpu provided is T4 or P100, one of the two GPUs will be randomly allocated. The amount of compute using GPU in the pro version is significantly increased. Pretrain

masked language modeling can last up to nearly 24 hours of continuous training. However, a limitation of google colab is that sessions will be interrupted after 24 hours, which means that if you do not back up data and checkpoints in time, all will be erased. Kaggle notebook also supports gpu like google colab but interacting with files to edit the code inside is not as easy as colab. Kaggle notebook regularly checks whether the user is still active on the notebook on a random basis from 3 to 4 hours, this is a big limitation compared to the colab pro version, the user must constantly interact with the kaggle notebook if you don't want to be interrupted. connect. Google Colab when disconnecting, if reconnect in a short time, the data is not deleted, kaggle is not. Kaggle has a GPU limit of 30 hours a week, Kaggle offers options such as 1 GPU P100 or 2 GPUs T4. Can be used to train models with multi-gpu techniques such as distributed data, data parallel or model parallel.

Google Colaboratory Google Colaboratory or Google Colab is a free cloud-based service provided by Google that allows users to run Jupyter notebooks on virtual machines (VMs) with access to GPUs, TPUs, and other hardware accelerators. It is designed to make machine learning and data science more accessible to everyone, particularly students and researchers who may not have access to high-end hardware or expensive software licenses. One of the main benefits of using Google Colab is that it provides free access to powerful hardware resources that can be used to train machine learning models and run data analysis tasks. The service also allows users to easily share and collaborate on Jupyter notebooks with others. However, there are some limitations to using Google Colab that users should be aware of: Limited Runtime: Colab provides virtual machines with a limited amount of computing resources. The free version of Colab provides 12 hours of continuous usage per session, and the session is automatically disconnected after 90 minutes of inactivity. If we let my program running and does not interact with the site, Google Colab still shut down, in the worst case, our checkpoints are thrown away, there is nothing left and we need to start again. It is very risky and spend a lot of time. Limited GPU and TPU Availability: While Colab does offer access to GPUs and TPUs, the number of available resources is limited and they may not always be available. Users may need to wait in a queue to access these resources. In order to train a high capacity model, in our case, we need to train more than 5 models to complete our projects, moreover, we need to re-build model for many experiments on different hyper-parameters. It is possible when working with limited computing units and runtime. We spend about 5.000.000 VND to buy more computing unit and service). Limited Storage: Colab provides a limited amount of storage space (approximately 68 GB) that is shared across all of a user's notebooks. Users may need to download and upload large datasets or model files between their local machine and Colab. No Guaranteed Uptime: Since Colab is a free service, there is no guarantee that it will be available or operational at all times. Users should always save their work frequently and have a backup plan in case the service becomes unavailable. Security: Colab is a shared environment, and

it is important for users to be aware that their code and data may be accessible to other users who have access to the same virtual machine. If we use free version, we are supplied 1x T4 GPU. For pro version, we also have 1x T4 GPU but 24 hours runtime actively. For pro plus verion, we get 1x GPU A100 but it is allowed to use a limited time until runing out of computing units, then get back to NVIDIA T4. Despite these limitations, Google Colab is a powerful and convenient tool for data scientists and machine learning practitioners. It offers a way to access powerful hardware resources and collaborate with others on Jupyter notebooks, all without requiring expensive hardware or software licenses.

Kaggle Notebook Kaggle is an online platform that provides a community for data science practitioners to collaborate, share ideas, and compete in machine learning competitions. Kaggle offers various features such as datasets, kernels, discussions, and competitions to its community. Kaggle Notebooks is one of the features that enables users to run Jupyter notebooks directly on the Kaggle platform. Kaggle Notebooks offers the following benefits: Easy Collaboration: Users can easily share their notebooks with others, allowing for seamless collaboration and knowledge sharing within the Kaggle community. Free Cloud-based Computing: Kaggle provides users with free cloud-based computing resources that include GPUs and TPUs, allowing for the training of complex machine learning models. Pre-installed Libraries: Kaggle Notebooks come with pre-installed libraries such as Pandas, Numpy, and Scikit-learn, making it easy for users to start working on their projects without needing to install these libraries themselves. Version Control: Kaggle Notebooks have built-in version control, allowing users to track and revert changes to their code. Reproducibility: Kaggle Notebooks allow users to easily reproduce their results by sharing the code and data used in their analyses.However, there are also some limitations to using Kaggle Notebooks: Limited Resources: Although Kaggle provides free cloud-based computing resources, the resources are limited, and users may need to compete for access during peak usage times. No Persistent Storage: Kaggle Notebooks do not provide persistent storage, which means that users may need to download and upload datasets and trained models between their local machine and Kaggle. Security: Kaggle Notebooks are a shared environment, which means that users should take appropriate measures to protect their data and code. In summary, Kaggle Notebooks is a powerful tool for data science practitioners looking to collaborate, share ideas, and compete in machine learning competitions. It provides free cloud-based computing resources, preinstalled libraries, and version control, making it easy for users to get started on their projects. However, users should be aware of the limitations of the platform, including limited resources and the need to manage their data and code carefully.

FPT Cloud FPT Smart Cloud (FCI) – a member of FPT Corporation, the leading provider of Artificial Intelligence (AI) Cloud Computing (Cloud Com-

puting) application solutions in Vietnam. FPT Smart Cloud was established with the mission of turning every business into a technology enterprise, with innovative breakthroughs in technology and products. FPT Smart Cloud aims to be the leading provider of Cloud Computing and AI thanks to its solid technology foundation, diverse product ecosystem and global connectivity. We rent a GPU server with 1x GPU A30, 300GB storage and 8x VCPU in order to train our project. It takes us about 12.000.000 VND for a month. The cost is shared by team members (two members). The NVIDIA A30 is a powerful data center GPU designed for mainstream AI and high-performance computing workloads. It is built on NVIDIA's Ampere architecture, which is known for its high performance and power efficiency. The A30 GPU is optimized for inference workloads and can deliver up to 10X higher inference throughput than CPU-only servers. Compared to the P100, which is based on the Pascal architecture, the A30 delivers up to 2.5X higher inference throughput and up to 2X higher memory bandwidth. The P100 was released in 2016 and is still widely used in data centers for training and inference workloads. Compared to the A100, which is NVIDIA's flagship data center GPU also based on the Ampere architecture, the A30 has fewer Tensor Cores and less memory bandwidth. The A100 is optimized for both training and inference workloads and can deliver up to 6X higher performance than the P100 for training workloads. Compared to the T4, which is also optimized for inference workloads, the A30 has more Tensor Cores and higher memory bandwidth. The T4 was released in 2018 and is popular for inference workloads in data centers and edge devices. In summary, the NVIDIA A30 is a powerful data center GPU optimized for inference workloads. It offers higher performance and memory bandwidth than the P100 and T4, but has fewer Tensor Cores and less memory bandwidth than the flagship A100 GPU.

4.1.4 Project Management Tool

Notion The Notion online application was created by Notion Labs Inc. and is a freemium productivity and note-taking tool. It provides administrative capabilities including bookmarking, task management, project monitoring, to-do lists, and more. Applications for desktop and mobile devices running Windows, macOS, Android, and iOS provide additional offline features. Custom templates can be made, movies and web content can be embedded, and real-time collaboration is possible. Kanban boards, tasks, wikis, and databases are all integrated into the collaboration platform Notion, which supports modified Markdown. For taking notes, managing information and data, and managing projects and tasks, it serves as a single workspace. Users can comment on ongoing projects, take part in discussions, and get feedback using this file management tool's single workspace. Cross-platform applications and the majority of web browsers can access it. A tool for "clipping" content from websites is included. It assists users in organizing their work, managing files, setting reminders, keeping agendas, and scheduling tasks. Equations can be written in block or inline form with the help of LaTeX support. To utilize Notion, no specialized training is needed. With the Notion AI functionality, users may produce and update content, summa-

rize existing notes, conduct daily standups, change the tone, translate, or check material. It also includes AI capabilities and a library of free and fee-based templates. For their Business and Enterprise tiers, security features include single sign-on with Security Assertion Markup Language and private team areas. SaaS tools including GitHub, GitLab, Zoom, Lucid Software, Cisco Webex, and Typeform are all integrated with Notion.

4.2 Methods

Vietnamese Legal Text Retrieval is developed mainly with Retriever module with the input is a question in string representation. There is a knowledge base, in this case, the knowledge base consists of legal documents. The question will be converted to embedding vector representation and compared to all of embeddings in knowledge base (each data text has its own corresponding embedding as well) by compute similarity scores such as dot product, cosine similarity, euclidean distance and so on. Relevant legal documents will be returned with a long text or just a title of the circulars and decrees. In practice, users need to lookup the legal answer in a convenient way, returning the titles is implicit and hard to be enjoyable, returning a full text of circulars and decrees contains redundant information which is not necessary to answer the user's question. Therefore, a question answering model is created to extract the main idea of the returned long circulars and decrees text. Figure 6 shows that this project will focus on pretraining language model in in-domain with masked language modeling due to lack of vietnamese legal text retrieval for training model explained in 4.2.2 and in 4.2.1 will show how to initialize training dataset for training Retriever module. Instead of using bert-based models or any transformer encoder-based models to build Sentence Transformers, which is outstanding in semantic textual similarity task explained in 4.2.5. This project will use Condenser and CoCondenser, which are explained in 4.2.3 and 4.2.4 to improve the performance of any transformer encoder-based model.

4.2.1 Processing data

The search engine frequently uses the ranking algorithm BM25, a paradigm for lexical matching to determine the degree to which a group of documents are relevant to a given question. Using the search terms that show in each record, it ranks the documents, creates negative sentence pairs for training Sentence Transformer in 4.2.5, these negative training samples are the most relevant articles to the query. In other words, although they are similar to the label, they might be not correct label. Data is divided into positive and negative sample in order to train Contrastive loss [31], which pull correct relevant sentence close to a anchor and push wrong samples away. Positive sample is from the data answer in the dataset (documents that match a given query). Negative sample is created by using BM25 to extract 50 or 20 sentences close well to a given query. Before being fed into the lexical matching algorithm, the data is preprocessed. Preprocessing techniques include word segmentation with Pyvi and

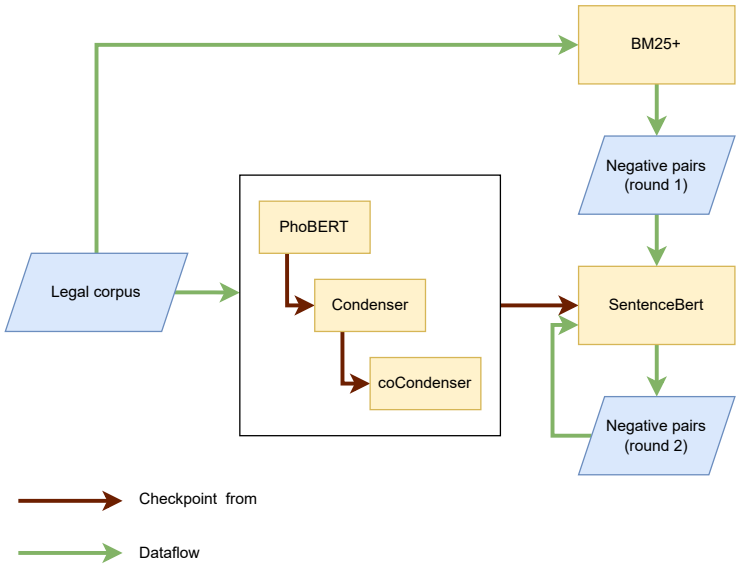


Figure 6: Training flow

text conversion to uppercase. The Retriever module is trained with 2 rounds based on [32]. Round one uses negative pairs extracted by using BM25 which is explained earlier to train Sentence Transformer with Contrastive loss. Round two uses Sentence Transformer trained in Round one in order to predict False Negative samples and continue to train Contrastive loss again on new data.

4.2.2 Pretrain Masked Language Model

In Masked Language Modelling, a certain percentage of the words in a sentence are typically hidden. These are masked by a special token, for example, [MASK], and the model is then supposed to predict the hidden words based on the other words in the sentence. In order to determine which word is filled up the masked places, we need to understand the context by observing the word's left and right side. Model is expected to perform this action when being applied this training approach, make the model bidirectional in nature. This can alternatively be thought of as a problem statement with blanks to be filled in.

Figure 7 represents how masked language modeling works. Fifteen percent of words in a sentence are chosen to be masked. Model needs to learn how to choose a correct word to replace each masked token in the input sentence. When being passed to several encoder block, the embedding vectors that represents for masked token, after being refined by learning context bidirectionally, embedding vector of each masked token is going through feed forward neural network and softmax layer to determined which word in vocabulary is a good candidate for this masked position based on probability. Given a set of N input token

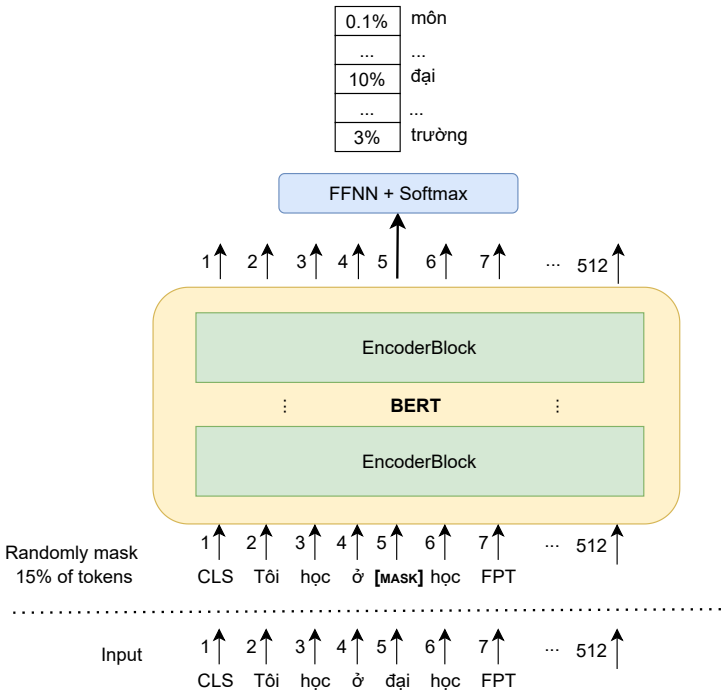


Figure 7: Masked Language Modeling examples

$T = t_1, t_2, ..., t_N$. Model masks out token t_i at position i and i is stored in set M which contains the index of masked token in T. The loss function of this task uses cross-entropy loss:

$$\hat{y}_i = W h_i^L \tag{22}$$

$$\mathcal{L}_{mlm} = \Sigma_{i \in M} \text{CrossEntropy}(\hat{y}_i, t_i) \tag{23}$$

where h_i^L is the embedding vector that represents for the masked token at position i at L-layer. $\hat{y}_i$ is predicted result after the masked token's embedding passed through feed forward neural network layer, W indicates the learnable layer.

Backbone language model we use is PhoBERT [2], a vietnamese encoder model. PhoBERT is trained on open-domain dataset, for example, vietnamese wikipedia so that it may not perform well in a specific domain including legal domain. In or der to improve the performance to encode semantic of legal documents conrrectly, we reference pretraining the language model way follow [3] that finetuning RoBERTa on 4GB of legal text data. We finetuning PhoBERT on masked language modeling task. However, we create 2 versions: PhoBERT finetuned on 300MB and 3GB of legal data. Because we do not have resource to train model with big dataset, we choose to suit our capabilities. The result is explained at 5 in details. Dataset for finetuning is collected the same with

[3] but less than it. By doing this way, we map the an open-domain phoBERT into an in-domain legal language model. This is exactly useful when we can not pretrain again a full phoBERT in a large legal dataset. After that, the new legal PhoBERT is used to train condenser in the next section.

4.2.3 Pretrain Condenser

Transfrormer model is used in semantic textual similarity takes [CLS] token, a representative of all word embedding in a sequence. However, [1] show that each token in the sequence, including the [CLS], only pays heed when receiving information from other tokens. The effectiveness of [CLS]'s knowledge aggregation is thus determined by attention patterns. According to [33], the author helps us to comprehend CLS's attentive behaviors: the CLS token is not attended by other tokens in the majority of middle layers, and it exhibits comparable attention patterns to other text tokens. Up until the very last layer, CLS has a singular, all-encompassing focus on the complete sequence to carry out NSP task, the CLS token is inactive in many intermediate layers and only becomes active during the final round of attention.

To solve this inactive behavior of regular transformer encoder model, Condenser is introduced with a bit different on top of an transformer encoder model. A regular transformer encoder model is divided into three group: early layers, late layers and head layers.

$$H_e = f_{ee}([h^0_{cls}; h^0]) \tag{24}$$

$$H_l = f_{le}(H_e) \tag{25}$$

where $f_{ee}(.)$ and $f_{le}(.)$ are encoder layers at early and late layers respectively. H_e and H_l is hidden state of the input sequence including [CLS] and other tokens at early and late layers in order. For traditional transformer encoder models, the input goes through each encoder layers as ususal, in this case, the hidden state h of each token is calculated by BERT normally, Eq. 24 and 24 show the computation. The difference is that hidden state of [CLS] and other tokens in the input sequence are still cached or stored in order to train head layers, which is the key of condenser architecture:

$$H_{head} = f_{head}([h^{late}_{CLS}; h^{early}]) \tag{26}$$

Input for condenser head $f_{head}(.)$ is taken from the hidden state of [CLS] output from late layers and other tokens output from early layers. This not only solve the inactive behavior of [CLS] token mentioned earlier, it still utilize the best [CLS]'s state of performance, but also reuse the information of other tokens at middle layers. H_{head} represents the last hidden state of each token output from the head layers including [CLS] and other tokens. In other words, H_{head} can be decomposed into $H_{head} = [h^{head}_{cls}, h^{head}_1, ..., h^{head}_N]$

The head layers are trained with MLM loss with the head's output presented in Eq. 27 and Eq. 29 which are re-written from Eq. 22 and Eq. 23 as follow:

$$\hat{y}^{head}_i = W h^{head}_i \tag{27}$$

$$\mathcal{L}_{mlm}^{condenser} = \Sigma_{i \in M} \text{CrossEntropy}(\hat{y}_i^{head}, t_i) \tag{28}$$

This new architecture design utilizes the meaningful CLS embedding at last layer of an encoder model, also avoid lack of information of other words by take them from the middle layers. In condenser, it tries to trains boost CLS hidden state more powerful. Despite being able to improve token forms, the late encoder backbone can only transmit new information through late CLS hidden state, make sure the backbone must aggregate freshly generated information later, and the head must condition on the late CLS representation to make LM predictions. By linking the early layers to the late layer, CLS is free from having to encode local information and the incoming text's syntactic structure, allowing CLS to concentrate on the overall meaning of the text. This informational division is controlled by the number of early and late layers.

Head layer helps condenser boost it performance in encoding meaningful of a sentence as much as possible. However, they are just exists in pretraining phase and are dropped in finetuning. When finetuning condenser in Semantic Textual Similarity or any relevant task, hidden state of CLS at late layer are powered up in pretraining phase, now is finetuned in a specific task, it is trained and backpropagate gradient into backbone. In the other words, condenser acts as a regular transformer encoder model in finetuning, it still have the same architecture with Transformer. The difference is that condenser has another step to power up the performance of CLS, representative token of a sentence by pretraining the LM backbone again added some layers, and the model is back to transformer architecture in finetuning and inference.

In this project, we chose to initialize Condenser using PhoBERT that has already pretrained in-domain with legal corpus and initialize the head arbitrarily. It means there are 12 encoder layers in our backbone PhoBERT, it is divided into the first 6 layers is early layers, the later 6 layers is late layers and create more 2 head layers. By eliminating the enormous cost of starting from scratch with pre-training, This fits within our computing budget. The result will be used to train Sentence BERT which is explained in Section 4.2.5. We impose a semantic restriction by running MLM also with backbone late outputs to stop gradient back propagated from the random head from distorting backbone weights.

$$\mathcal{L}_{mlm}^{constrain} = \Sigma_{i \in M} \text{CrossEntropy}(W h_i^{late}, t_i) \tag{29}$$

This restriction is justified by the assumption that encoding per-token representations h_i^{late} with $i \neq 0$ and sequence representation h_0^{late} where $i = 0$ is the position of [CLS] token have a comparable method and won't conflict. h_i^{late} with $i \neq 0$ is therefore still applicable for LM prediction. Therefore, the total loss is determined as the sum of two MLM losses.

$$\mathcal{L}^{total} = \mathcal{L}_{mlm}^{condenser} + \mathcal{L}_{mlm}^{constrain} \tag{30}$$

The two MLM losses share the output projection matrix W, which lowers the overall number of parameters and memory requirements.

4.2.4 Pretrain CoCondenser

Pre-trained language models, specifically encoder models, are often used to fine-tune and create dense retrieval models. However, training a dense retrieval model is not easy, requiring very high machine resources to train, clean data sources and huge human resources to process datasets, ensuring training. dataset in the most suitable state for a dense retrieval problem. Two common problems that directly affect the performane of dense retrieval models are noise and large batchsize during training to strongly enhance the model's ability to learn to distinguish contexts, in order to generate a vector. embedding represents a quality sentence and carries the most complete semantics. Model RocketQA [34] is one of the typical models that does both of those things to produce a quality dense retrieval model. This method offers improvements such as the removal of many hard negative pairs when training dual-encoder with contrastive loss. More specifically, in the process of labeling the retrieval datasets, the tagger will often label the missing passages, resulting in passages that should have been included in the question's list label being hard negative. leading to model noise. (Example when testing the question of what is a service flight, there are many circulars and decrees that all explain what a service flight is but the label has only 1, this leads to other circulars and decrees being false, while the selected label is true even though they have the same meaning, explaining the same concept). The second thing that RocketQA proposes to improve is to increase batchsize training, RocketQA trains in-batch negative, they have the problem that, during the inference process, the query has to compare with millions of data in the database. data, while in the training process, the model only learns to distinguish the query with a small number of negative samples. RocketQA trained dual encoder model with large batchsize to solve this problem.

However, the whole flow of the method is not really suitable for most of those who want to train dense retrieval model with limited resources, not everyone has a machine that is good enough to train large batch sizes, and has financial resources. or human resources to denoise labels that are missing by the labeler. [35] has provided a solution for this situation. They agree that removing the noise problem in the dataset will improve the model quality because most language models are very sensitive to noisy labels, the parameters for the model will be updated incorrectly. Regarding the problem of retrieving the query when it has to model to distinguish a large amount of knowledge to choose the one that best fits the query. , or is not explicitly trained, thereby leading to the inability to create a good representative vector, carrying full information and semantics for the whole sentence. [35] will train the backbone language model so that it has local anti-interference ability, that is, self-resolving noise problems, anti-interference during training and weight updating, creating a representative vector that CLS is explicitly pretrained, and trained on a sensible architecture. Based on that develop dense retrieval.

For noise resistant, they take advantage of the language model pretraining activity carried out by the Condenser pre-training architecture, explained in

Section 4.2.3, which is actively conditioned on the CLS vector. It creates a CLS representation that is information-rich and capable of condensing an input sequence with reliability. Then, they present a straightforward corpus-level contrastive learning objective: given a target corpus of documents from which to retrieve, sample text span pairs from a batch of documents at each training step, and train the model so that the CLS embeddings of two spans from the same document are close and those of spans from different documents are far apart. Combining the above two factors, the proposed coCondeser pre-train technique enables the model to perceive a wider range of observation, corpus-aware in unsupervised manner.

The authors of a research paper explain that although a machine learning model called "Condenser" can be trained on different types of information to create a universal understanding, it still has a problem with understanding the meaning of the information it learns. Even though the model can interpret some of the information, the relationships between the different pieces of information still lack meaning, so they don't work well together. To fix this problem, the authors added a new technique called a "contrastive loss" to Condenser. They also suggest training Condenser to understand different documents in a more general way, rather than training it on specific question-answer pairs. They do this by randomly selecting pairs of information from different documents to help Condenser learn more generally. The specific part of the information that Condenser learns from is called a "span," and the contrastive loss helps Condenser better understand the meaning of these spans. The contrastive loss, which takes into account the corpus, is defined for the entire training batch:

$$\mathcal{L}_{ij}^{co} = -\log \frac{\exp([h_{i1}; h_{i2}])}{\sum_{k=1}^{n} \sum_{l=1}^{2} \mathbb{I}_{ij \neq kl} \exp([h_{ij}; h_{kl}])} \tag{31}$$

This equation defines the corpus-aware contrastive loss function, denoted by $\mathcal{L}_{ij}^{co}$, for pre-training the passage embedding space using the coCondenser model. The loss function measures how well the model is able to distinguish between pairs of spans within the same document that should be semantically close, and pairs of spans that should be far apart. The loss function is defined in terms of the late CLS representations of two spans, h_{i1} and h_{i2}, which are concatenated together to form a single vector. The numerator of the fraction computes the similarity between the two spans using the dot product of their concatenated representations, which is transformed using the exponential function. The denominator sums the similarity between the two spans and all other spans in the batch, with the exception of the same span paired with itself or its inverse. The $\mathcal{I}_{ij \neq kl}$ term is an indicator function that is equal to 1 when the indices i, j, k, l satisfy the inequality $ij \neq kl$, and 0 otherwise. In essence, the loss function encourages the model to learn embeddings that encode the semantic meaning of each span, such that similar spans are assigned similar embeddings, and dissimilar spans are assigned dissimilar embeddings. This is accomplished by minimizing the negative logarithm of the similarity between two spans divided by the sum of similarities between all pairs of spans in the batch, excluding pairs that are semantically identical. The contrastive loss used in the equation is inspired by the

contrastive loss used in SimCLR [36], where the goal is to learn representations by contrasting positive and negative pairs of augmented examples. In this case, the positive pairs are the two spans from the same document, and the negative pairs are the spans from different documents. The use of the contrastive loss is a form of noise contrastive estimation (NCE), which is a technique used to estimate the probability distribution of a random variable by contrasting it with a noise distribution.

Noise Contrastive Estimation (NCE) is a technique for estimating the probability of a certain event or occurrence, based on a limited set of observations. This is often used in machine learning and natural language processing, where we want to estimate the probability of a certain word or phrase appearing in a given context. The basic idea behind NCE is to train a model to distinguish between "real" and "fake" examples of the event or occurrence we are interested in. The model is given a set of "real" examples, and a larger set of "fake" examples, which are generated by adding some random noise to the real examples. The model is then trained to predict whether a given example is real or fake, based on its features. For example, let's say we want to estimate the probability of the word "cat" appearing in a sentence. We could train a model using NCE by giving it a set of real examples, which are sentences that contain the word "cat", and a larger set of fake examples, which are sentences that don't contain the word "cat", but are generated by randomly adding or changing some words in the real examples. The model is then trained to distinguish between real and fake sentences, based on their features (e.g. the presence or absence of certain words). Once the model is trained, we can use it to estimate the probability of the word "cat" appearing in a new sentence. We simply give the model the features of the new sentence, and it outputs a probability score, indicating how likely it is that the sentence contains the word "cat". he main difference between NCE and training with hard negative or in-batch negative when using contrastive learning is in how the negative samples are chosen. In hard negative mining, the negative samples are chosen to be the hardest ones to classify correctly among a set of randomly selected candidates. In other words, the algorithm selects samples that are most similar to the positive sample, but are still labeled as negative. In-batch negative mining, on the other hand, selects negative samples from within the same batch as the positive sample. This means that the negative samples are taken from the same set of data that the model is currently training on. In contrast, NCE selects negative samples from a noise distribution that is different from the training data. The noise distribution is designed to be easy to sample from, but has a different distribution than the training data. By sampling negative examples from the noise distribution, the model is forced to learn to discriminate between the true training data and the noise distribution. This can help prevent the model from overfitting to the training data and can lead to more generalizable representations. Here's an example to help illustrate the difference between the three approaches: Suppose we have a dataset of images of cats and dogs, and we want to train a model to classify them correctly. With hard negative mining, we would randomly select a negative sample, and then choose the sample that is most similar to the positive

sample, but still labeled as negative. With in-batch negative mining, we would select a negative sample from within the same batch of images that the positive sample is in. With NCE, we would sample negative examples from a noise distribution, such as a distribution of random noise images or a distribution of images of objects that are not cats or dogs.

The authors of the coCondenser paper noted that the contrastive loss used in their approach is similar to the one used in NCE, where the loss function aims to distinguish between true samples (i.e., samples from the actual data distribution) and noise samples (i.e., samples generated from a noise distribution). In the context of coCondenser, the authors use random span sampling as a form of noise contrastive estimation, where random spans from the input documents are used as negative samples during training. The contrastive loss function is then used to encourage the model to differentiate between true positive samples (i.e., spans that are semantically related) and negative samples (i.e., random spans that are not semantically related). Therefore, the coCondenser approach can be seen as providing an NCE narrative, where the noise samples are generated through random span sampling, and the contrastive loss is used to encourage the model to distinguish between positive and negative samples. Let's say we have a document that talks about the benefits of exercise, and we want to use coCondenser to generate embeddings for the different spans of text in this document. For instance, one span could be "running is good for your heart" and another span could be "lifting weights can help build muscle". These two spans are semantically related, since they both talk about the benefits of different types of exercise. On the other hand, a random span in the same document that is not semantically related could be "the sky is blue". This span is not related to the topic of exercise and would not be used in the training process. During training, coCondenser takes pairs of spans from the same document and tries to learn to distinguish between semantically related and unrelated spans using the contrastive loss. This helps it to generate embeddings that capture the semantic meaning of the text. The coCondenser model assumes that spans that are not semantically related have a low probability of being selected in a random sampling process. This is because the model uses a corpus-aware contrastive loss that compares the similarity between the representations of different spans across the entire corpus. In other words, if two spans are not semantically related, they are likely to have different representations across the corpus, and therefore the contrastive loss will penalize the model for treating them as similar. Of course, this assumption is not perfect, and it is possible that two spans that are not semantically related may have similar representations by chance. However, the coCondenser model is designed to minimize the impact of these cases by using a large corpus and random sampling of spans. The training data for coCondenser is sampled from spans taken from different documents in a corpus. The purpose of sampling from a corpus is to ensure that the model learns to represent information in a generalizable way rather than just memorizing specific instances in the training data. So, although the data used to train coCondenser comes from a corpus, it is not trained on the entire corpus at once, and the training data is sampled in a way that promotes generalization.

the authors are explaining the approach they have taken in training their coCondenser model. They start by stating that they use random spans as surrogates of passages, which means that they randomly select spans of text from different documents as their training data. They then enforce the distributional hypothesis through noise contrastive estimation (NCE), which is a method commonly used in word embedding learning (e.g., Word2Vec) to learn representations of words based on their co-occurrence patterns in a corpus. By using NCE, the authors aim to ensure that the model learns to distinguish between semantically related and unrelated spans. They then go on to explain that this approach can also be seen as a span-level language model objective, similar to the popular "skip-gram" model used in word embedding learning. Finally, they state that the batch's loss is defined as an average sum of MLM and contrastive loss, which can also be seen as word and span LM loss, respectively. The authors of coCondenser are drawing a comparison between their proposed span-level language model objective and the popular "skip-gram" model used in word embedding learning, such as in Word2Vec (Mikolov et al., 2013). In the "skip-gram" model, the goal is to learn word embeddings by predicting the context words surrounding a target word within a fixed window size. This can be seen as a type of language modeling, where the target word is the input and the context words are the output. Similarly, in coCondenser, the authors propose to use spans as surrogates of passages and enforce the distributional hypothesis through NCE, which is a form of contrastive learning. The MLM loss of the span-level language model is similar to the word-level language model in "skip-gram", and the contrastive loss helps to further refine the embedding space by distinguishing related spans from unrelated ones. So, by making this comparison to "skip-gram", the authors are highlighting the similarity of their proposed approach to the widely-used technique in word embedding learning.

The authors of the coCondenser paper use two types of losses to train their model: MLM loss and contrastive loss. The MLM loss is a standard loss used in many language models, which aims to predict the masked tokens in a sequence based on the context provided by the other tokens. In coCondenser, the MLM loss is computed for each span separately, denoted as $\mathcal{L}_{ij}^{\text{mlm}}$ for span s_{ij}. The contrastive loss, as we discussed earlier, aims to distinguish between semantically related spans and randomly selected spans. The contrastive loss is computed based on pairs of spans selected from the training corpus, and it penalizes the model if the similarity between a semantically related pair is lower than that of a randomly selected pair. The contrastive loss for a pair of spans is denoted as $\mathcal{L}_{ij}^{co}$. To combine the two losses, the authors define the batch's loss as an average sum of the MLM loss and the contrastive loss. That is, for a batch of spans, the total loss is calculated as follows:

$$\mathcal{L} = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{2} [\mathcal{L}_{ij}^{\text{mlm}} + \mathcal{L}_{ij}^{co}] \tag{32}$$

In other words, the batch's loss is a weighted average of the MLM loss for each individual span in the batch, and the contrastive loss for pairs of spans in the

batch. This loss function encourages the coCondenser model to learn both the semantic relationships between spans and the contextual information within each span.

In the paper, the authors refer to "unsupervised factors" as the underlying semantic structure that is present in unannotated text data. These factors capture the relationships between words and phrases in the language, such as synonyms, antonyms, and related concepts, and can be learned by a language model trained on a large corpus of text data. The coCondenser model is designed to capture these unsupervised factors by using a contrastive loss to learn to distinguish between related and unrelated spans of text. By doing so, it can create a more effective embedding space that can be used in downstream natural language processing tasks.

Stochastic gradient estimators (SGEs) are methods used to estimate the gradient of the loss function with respect to the model parameters using only a subset (or a single example) of the training data at each iteration. The idea is to randomly sample a subset of the training data, called a mini-batch, to compute an approximate gradient and update the model parameters. This process is repeated for multiple iterations until the model converges to a minimum of the loss function. Gradient estimators are algorithms used to estimate the gradient of a function with respect to its parameters. In machine learning, the gradient is used to update the model parameters during training to minimize the loss function. There are several gradient estimators, including the stochastic gradient estimator, which is commonly used in deep learning. The authors of the coCondenser paper mention "large-batch unsupervised pretraining" as a way to construct effective stochastic gradient estimators for the contrastive loss. This means that they first pretrain their model on a large dataset using unsupervised methods, without any specific task or objective in mind. The purpose of this pretraining is to teach the model to understand the structure and patterns of language in a general sense, so that it can later be applied to specific tasks more effectively. Once the pretraining is done, the model is then fine-tuned on a specific task or dataset, such as a question-answering task or a language generation task. The fine-tuning is done using small batches of data, which allows for more efficient training and better generalization to new data. The authors argue that the large-batch unsupervised pretraining step is crucial for achieving good performance on downstream tasks, because it helps the model learn useful representations of language that can be applied to a wide range of tasks. They also suggest that this pretraining step can be done once on a large dataset, and the resulting model can be reused or adapted for different downstream tasks, which saves time and resources compared to training a new model from scratch for each task. the contrastive loss requires fitting the large batch into GPU memory, which can be challenging with limited resources, such as a machine with only four commercial GPUs. To overcome this memory constraint and perform effective contrastive learning, they incorporate a technique called gradient caching [cite]. Gradient caching involves storing intermediate gradient values for the parameters during training, allowing the system to perform more efficient backpropagation during subsequent epochs.

This reduces the memory usage during each batch and allows the system to process larger batches, even with limited GPU resources. In essence, gradient caching allows the system to approximate the gradient of the large batch with a series of smaller batches, without losing accuracy or incurring a significant computational overhead. Gradient caching and gradient checkpointing are both techniques used to reduce the memory requirements of deep learning models during training. Gradient caching refers to storing intermediate computations of gradients during the forward and backward passes of the model, rather than recomputing them during each iteration. This can reduce the amount of memory required during training, but can also result in slower training times due to the additional overhead of caching. Gradient checkpointing, on the other hand, involves recomputing intermediate activations during the backward pass, rather than storing them in memory. This can reduce the memory requirements of the model even further, at the cost of additional computation time. In the context of the coCondenser paper, the authors use a variant of gradient caching called "recompute-aware gradient caching". This involves recomputing intermediate activations during the backward pass, but also caching a subset of activations in memory to reduce the overhead of recomputation. This allows the model to train effectively on machines with limited GPU memory.

Storing intermediate computations of gradients during the forward and backward passes of the model can reduce the amount of memory required during training because it allows the model to perform backpropagation and compute gradients in a more memory-efficient way. During backpropagation, the model computes gradients for each parameter by multiplying the gradient of the loss function with respect to the output of the layer with the gradient of the layer's output with respect to its input. These gradients can be quite large and storing them all in memory can quickly become unfeasible, especially for large models or when training on GPUs with limited memory. By caching intermediate computations of gradients, the model can free up memory by discarding unnecessary computations that would otherwise need to be stored. This can help reduce the memory footprint of the model and make it possible to train larger models or use larger batch sizes without running out of memory. Gradient checkpointing is one way to implement gradient caching, where intermediate activations are recomputed during the backward pass rather than stored in memory. This allows the model to use less memory during training at the cost of increased computational time. The memory referred to in this context is the GPU memory required to store intermediate computations of gradients during the training process. Storing these intermediate computations allows the model to compute gradients for larger batch sizes without running out of memory on the GPU. Recomputing intermediate activations during the backward pass can reduce the memory requirements of the model even further because it allows the model to discard the intermediate activations after they have been used in the backward pass. This means that the memory used to store the intermediate activations can be freed up and used for other purposes, such as storing activations for the forward pass or computing gradients. Without recomputing the intermediate activations during the backward pass, the model would need to store all the

intermediate activations for each layer until the gradients are computed for the final layer. This can be very memory-intensive, especially for large models or models with many layers. By recomputing the intermediate activations during the backward pass, the model can avoid storing these intermediate activations, thereby reducing the overall memory requirements.

The pretraining of coCondenser is done in two stages:

- Universal Condenser pretraining: In this stage, a Condenser is pretrained using the same data as BERT, i.e., English Wikipedia and the BookCorpus. The Condenser is initialized with the pre-trained 12-layer BERTbase weights and its backbone layers are warm-started using an equal split of 6 early layers and 6 late layers. The pretraining objective is to learn a general-purpose representation of language that can be fine-tuned on different downstream tasks.

- Corpus aware coCondenser pretraining: In this stage, the pre-trained Condenser from stage one is taken and its backbone and head layers are used to warm-start the pretraining on the target corpus, which can be either Wikipedia or MS-MARCO web collection. The pretraining objective is to learn a corpus-specific representation of language that can be fine-tuned on specific downstream tasks related to the target corpus. The architecture of the Condenser is kept unchanged in this stage.

During both stages of pretraining, the coCondenser model uses the contrastive learning framework and NCE objective to learn the representations of spans sampled from the input corpus. The loss function used for pretraining is a combination of MLM loss and contrastive loss, with MLM loss being used to predict the original spans and contrastive loss being used to distinguish them from negative spans. The negative spans are sampled randomly from the same document as the original spans. The pretraining process is done in batches, with each batch consisting of multiple spans sampled from different documents. The gradients are calculated for each batch using backpropagation and used to update the model parameters. The pretraining process continues for a fixed number of epochs until the model converges and achieves the desired level of performance. After pretraining, the coCondenser model can be fine-tuned on specific downstream tasks by adding a task-specific head layer and training the entire model end-to-end using supervised learning.

4.2.5 Build Sentence Transformer

Transformer models use attention to refine the embedding representation of each words in the sequence by paying attention to the factors mainly affect to itself. However, it just work to create token-level embeddings, not sentence-level embeddings while we need our query input and documents in database represented as a embedding vector

In order to compare 2 sentences, cross-encoder structure was introduced by taking 2 sentences into a single BERT model, or any transformer encoder model.

On top of this structure, a feed forward neural network perform classify whether two sentences are similar by applying sigmoid or using any similarity score like cosine similarity.

Comparing each pair of sentences makes cross-encoder is not scalable although it produces very accurate similarity scores (better than the following Sentence BERT about to be mentioned). We would have to do the cross-encoder inference computation 100K times if we wanted to run a similarity search across a tiny dataset of 100K sentences. Because the input of cross-encoder is a pair of sentences separated by a special token, it is hard to use this model to generate semantic embedding vector for each document in advance and invoke them from database, compute similarity score when needed. Next, By averaging the values across all token embedding output, original BERT creates semantic sentence embeddings, it mean last hidden state of transformer encoder models. On the other hand, special token, CLS token which stands in front of every sequence when training with transformer encoder model as an alternative way to represent for the sequence and take it to comparison operator. But regardless of which method is used, the accuracy is poor and is worse than utilizing averaged GloVe embeddings.

Sentence-BERT, also known as SBERT, was developed as a remedy for this lack of an accurate model with a respectable latency. For every standard semantic textual similarity (STS) task, SBERT performs better than the prior state-of-the-art (SOTA) models. SBERT provides sentence embeddings, which is a blessing for scalability because it eliminates the requirement for a full inference calculation for every sentence-pair comparison.In 2019, Reimers and Gurevych presented evidence of the sharp acceleration. With BERT, it took 65 hours to identify the most analogous pair of sentences among 10K sentences. The creation of embeddings with SBERT takes around 5 seconds, while the cosine similarity comparison takes about 0.01 seconds.

Many more sentence transformer models have been developed since the SBERT paper using ideas that were used to train the original SBERT. They have all been practiced on numerous pairs of sentences, both similar and dissimilar. These models are optimized to create comparable embeddings for related sentences and dissimilar embeddings in all other cases using a loss function such as softmax loss, multiple negatives ranking loss, or MSE margin loss.

In this project, Sentence BERT with the backbone is legal Condenser explained in Section 4.2.4, called CoLegalPhoBERT, instead traidtional transformer encoder model such as BERT or RoBERTa in [37]. Our Sentence BERT using coCondenser and is finetuned by creating siamese network but trained with constrastive learning approach instead triplet networks in the original Sentence BERT paper. At the end of pre-training, the authors discard the Condenser head, which includes the final prediction layer, and keep only the backbone layers. As a result, the model reduces to its backbone, or effectively a Transformer Encoder. The weights of the backbone layers are then used to initialize the query encoder and passage encoder in the downstream task of retrieval. Specifically, the query encoder (Eq. 33) and document encoder (Eq. 34) are each initialized with the weights of the corresponding backbone layer outputting the

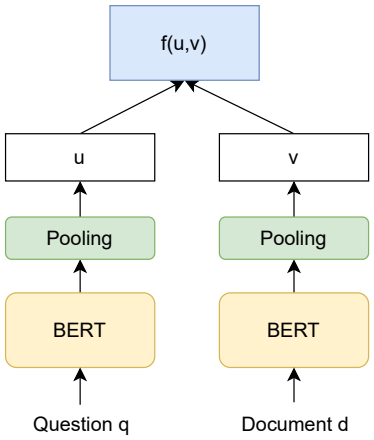


Figure 8: SBERT architecture with objective function

last token's CLS representation. This initialization process allows the retrieval model to benefit from the pre-trained knowledge captured by the coCondenser model. Figure 8 has shown overview SBERT architecture. The pooling layer which is used in this project is mean pooling layer. Objective function $f(u, v)$ is contrastive loss. Question and documents are encoded to semantic embedding vector by:

$$q = \text{CoLegalPhoBERT}([\text{CLS}; \text{question}; \text{SEP}]) \tag{33}$$

$$d = \text{CoLegalPhoBERT}([\text{CLS}; \text{document}; \text{SEP}]) \tag{34}$$

Question and Document are bounded with beginning special token [CLS] and ending special token [SEP] and passed into CoLegalPhoBERT, to create semantic embedding vector. In training, $f(u, v)$ in Figure 8 indicates $\mathcal{L}_{\lfloor}$ ranking loss which can be calculated with both positive document d_{pos} and negative documents d_{neg} for the given question to contrastively train CoLegalPhoBERT:

$$\mathcal{L}_{sb} = \Sigma_q \Sigma_{d_{pos} \in D_{pos}} l_c(q, d_{pos}, D_{neg}) \tag{35}$$

Where D_{pos} is the positive document collection for the given question q. $l_c(q, d_{pos}, D_{neg})$ is the contrastive loss function, which can be referenced from [38]:

$$l_c(q, d_{pos}, D_{neg}) = -\log \frac{e^{f(q_{pos}, d_{pos})}}{e^{f(q_{pos}, d_{pos})} + \Sigma_{d_{neg} \in D_{neg}} e^{f(q_{pos}, d_{neg})}} \tag{36}$$

Where D_{neg} is the collection of negative documents for given question q sampled with sparse retrieval methods, in this project we use BM25+.

4.2.6 Question Answering

Sentence Transformer returns the most relevant documents to a given query but they are all too long and some of returned information is not necessary.

Therefore, a question answering exists to extract relevant pieces of knowledge from context returned by Retriever Module.

There are two kinds of question answering: extractive and abstractive question answering. Extractive question answering is popular because there are a lot of datasets created for this task in a number of domain. Inputs of extractive question answering include question and relevant contexts. They are separated by a special token, in this case, phoBERT has [SEP] lied between question and context, the model will return two numbers: start and end position, it means the position of the start and end character in the answer span extracted from the context. This type of question answering task is more likely suitable for legal question answering model because it answers a legal question surely based on given knowledges, there is no changed or paraphrase the answer which is extremely sensitive in legal domain where accuracy of each word in a sentence counts.

The sencond type of question answering task is abstractive question answering, or generative question answering. Model will take the inputs that are the same with extractive question answering task but the output is a probability distribution of each word in model's vocabulary so that we can use some decoding algorithms such as greedy search, beam search, top-p and top-k sampling to generate human-like text answer. The answer is represented in the human-readable way. Although this makes the chatbot model acts as a real human, but in the legal domain, generation is hard to be committed that the generated answer is suitable and safety while there is no any criteria to control and handle the quality of a generative model. Therefore, extractive question answering is highly recommended in legal domain and it is also performed in this project. While encoder models are currently the preferred choice for extractive question answering, sequence-to-sequence (Seq2Seq) models can also have advantages in certain scenarios:

- Non-contiguous answer spans: Seq2Seq models are well-suited for tasks where the answer span is not a contiguous sequence of words. This is because they are capable of producing an output sequence that is not constrained to a fixed length.
- Additional context: Seq2Seq models can incorporate additional context beyond the given passage to answer a question. This can be useful for tasks that require a broader understanding of the topic or require reasoning beyond the information contained in the input passage.
- Language Generation: Seq2Seq models are capable of generating natural language output, which can be useful for tasks that require the system to generate an answer in a specific format or style.

Despite of performing extractive question answering, we do not output two numbers that are start position and end position. We notice that predicting is not associated to the idea of question answering task. We pass a query sentence and its relevant knowledge, then expect the model to return the answers existing

in one of given contexts. start position and end position are not originally the idea of using language model to create an answer, answer a question like human. Model is better to predict a span of text where each token is generated based on the previous one until complete the answer. Predicting the beginning of the supposed answer and its ending token, then slice the span of text is not appropriate. Proof is shown below:

5 EXPERIMENTS AND RESULTS

5.1 Dataset

Vietnamese Legal Text Retrieval The data we used for the "Vietnamese Legal Text Retrieval" section comes from the "Legal Text Retrieval" dataset in the "ZALO AI Challenge 2021", which includes up to 3200 legal articles. In addition, we collected legal data from Vietnamese sources such as "Lawnet.vn" and "vbpl.vn", with a total of nearly 145,000 legal documents spanning from October 2018 to January 2023. After processing steps to filter out noise and remove duplicate words in sentences, we obtained nearly 3GB of legal data.

Question Answering In order to address the problem of limited training data for legal question answering, the authors trained their legal Phobert model with UIT-ViQuAD [39], a collection of 23,000 questions and answers created by humans using passages from 174 Vietnamese Wikipedia entries. By doing this, the extractive question answering model can first learn reading comprehension skills before being applied or performed inference on the 520 legal questions and 1377 articles from the Automated Legal Question Answering Competition (ALQAC 2022). After checking 520 questions, we found that there are 9 questions contain the answer is not a pieces of information that can be extracted from a given question's corresponding articles. Because we trained our Vqa-ViT5 to perform extractive question answering task, we discard these unsuitable question and retain 511 samples

5.2 Processing data

Data processing section of collecting data from Vietnamese law websites, we follow the process of standardizing punctuation and using pre-built Vietnamese text forms.

The combined and pre-built Vietnamese character sets are created to support typing and displaying Vietnamese characters on computers. They include all the characters in the Vietnamese alphabet, including tone and punctuation marks.

- The combined set is named "combined" because it consists of characters created by combining basic characters in the Vietnamese alphabet. Other characters such as "ê", ... are also created by combining different basic characters.
- The pre-built Vietnamese character set is integrated into computer systems or application software, eliminating the need for users to install this character set. It is provided by software manufacturers and supported on most current operating systems and application software.
- Both the combined and pre-built Vietnamese character sets play an important role in supporting the use of Vietnamese on computers and mobile

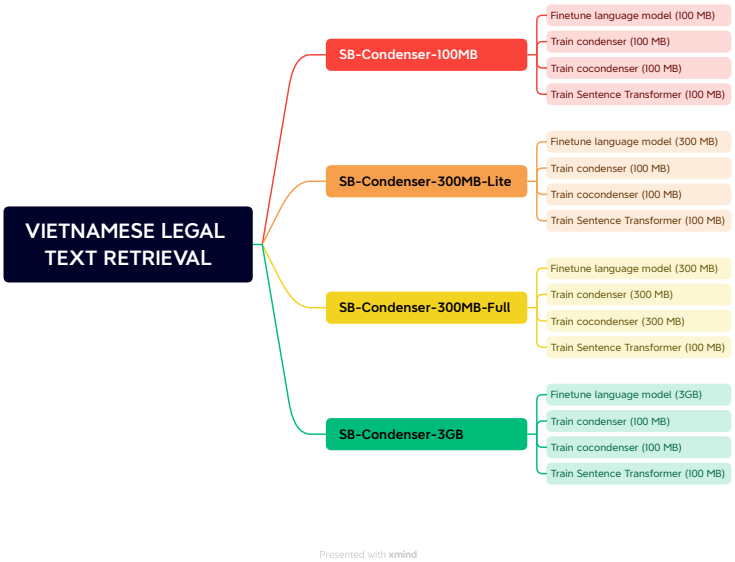


Figure 9: Overview about four versions in Vietnamese Legal Text Retrieval

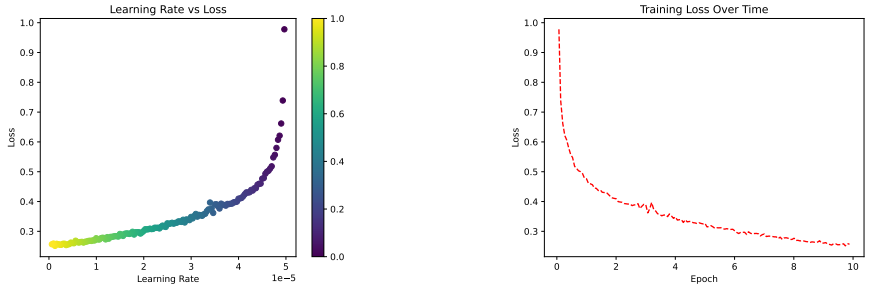
devices. However, users need to pay attention to using the correct characters and marks in Vietnamese to avoid confusion or misunderstanding the meaning of words.

In Vietnamese, punctuation marks play an important role in distinguishing the meanings of words. Incorrect use of punctuation can lead to misunderstanding or loss of the meaning of a sentence.

- For example, in the case of "hoàng" and "hòang", these are two different words in meaning. "hoàng" means a name, while "hòang" has no meaning in Vietnamese.
- Other punctuation errors in Vietnamese include not using the period, comma, or punctuation marks correctly, or placing them in the wrong position. For example, the difference between the sentences "Anh yêu em" and "Anh, yêu em" or "Anh yêu, em" can change the meaning of the sentence. Therefore, to avoid misunderstandings about meaning, using punctuation correctly is very important in Vietnamese.

5.3 Pretrain Masked Language Model

Describe presentation: We have a total of 4 experimental models, which are SB-Condenser-100MB, SB-Condenser-300MB-Lite, SB-Condenser-100MB-Full, and SB-Condenser-3GB, respectively. However, the same training parameters



(a) Learning Rate vs Loss

(b) Train loss over time

Figure 10: Overview about training loss

were used in the parameter setup. Therefore, throughout the process of describing the implementation method, we only represent the **SB-Condenser-300MB** model in the core main section.

5.3.1 Setting parameters of training model

Table 3: Training parameters of Pretrain Masked Language Model.

Model Name	Train Epochs	Total Batch Size
Phobert Large	10	32

In setting the parameters for fine-tuning the Phobert Large language model, we chose Epochs as 10 for training, and set the batch size for both the evaluation and training sets as 8. Additionally, we set a parameter to increase the batch size of the model to 32 by setting the Gradient Accumulation parameter to 4 (8*4). The reason we increased the batch size through Gradient Accumulation instead of directly increasing it is that directly increasing the batch size would lead to an increase in GPU memory usage. In this training session, we used two types of GPUs: T4 from Google Colab and P100 from Kaggle Notebook, both with GPU memory ranging from around 13GB. This means that we cannot increase the batch size further without encountering memory overflow issues. Therefore, setting Gradient Accumulation ensures that the batch size can increase up to a maximum of 32. Additionally, we also wanted to ensure that the batch size was not too low, as having too low of a batch size would result in issues with the loss function during training.

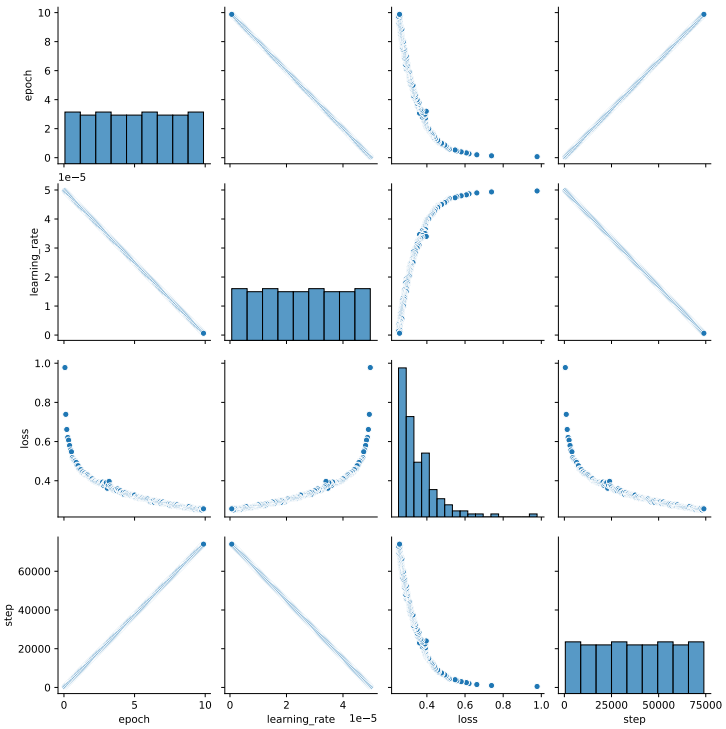


Figure 11: Overview about pretrain Masked Language Model

5.3.2 Visualize Results

Regarding Figure 10, it shows us detailed insights into the results of the loss function compared to other training parameters such as learning rate and epochs. In Figure 10a, the learning rate vs loss plot indicates that both parameters are directly proportional to each other throughout the training process. The learning rate values range from below $1e-5$ to $5e-5$, while the loss values range from below 1.0 to approximately 0.2. Moreover, we can clearly see that the automated parameter tuning process gradually adjusts the learning rate and monitors the loss function in parallel. The results demonstrate that the training process performs well in enabling the PhoBERT Large language model to learn new word connections and is particularly useful in the legal domain. In figure 10b, we can see another view of the loss parameter compared to epochs. It is clear that the loss decreases during the model training process, specifically when we set the epoch parameter to 10. In some other machine learning or deep learning problems, setting the epoch parameter to 10 or less may be too few and cannot guarantee optimal results in model training. However, in the field of natural language processing or fine-tuning for the Phobert Large language model in this case, with a considerable amount of data and the specificity of this language

model with over 340 million parameters, setting the epoch parameter to 10 can save time during the model training process while still ensuring similar good results as when using a larger epoch parameter. To clarify this, we can refer back to Figure 10b, where we observe a significant decrease in the loss function, especially during the first few epochs where the loss function reaches around 0.5, and then a slight decrease from 0.5 to almost 0.2 during the subsequent epochs. Moreover, overall, the loss function decreases during the entire training process with slight fluctuations and a notable decrease during the third epoch.

With regard of the Figure 11, we use the seaborn library to visualize four main parameters: step, learning rate, loss, and epoch. From all the charts in the figure, we can also have a deeper insight into the correlation and variation of these parameters over time during the model training process. We can also focus mainly on the learning rate parameter compared to other parameters, clearly the model can be evaluated as relatively good because it maintains the loss decreasing and not varying too much. Furthermore, by using dots to clearly represent the position of the losses over time, we can also see a slight variation from epoch 3 to epoch 4. Besides, there is also a bar chart comparing the loss with itself to clarify that it still decreases and illustrate the variation of this parameter more visually. Overall, setting the parameters for this language model training, specifically the Phobert Large model with 300MB of clean legal data and the training parameters mentioned above, resulted in a very objective training outcome and can ensure the quality for the next training modules such as Condenser, Cocondenser, and Round 1 and 2 of Sentence Transfer.

5.4 Pretrain Condenser

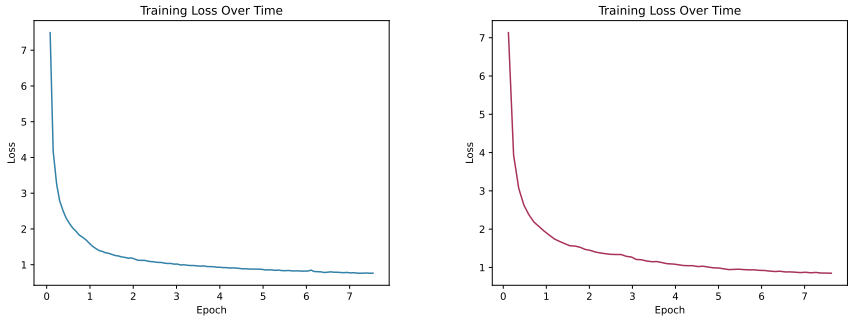
Describe presentation: We have a total of 4 experimental models, which are SB-Condenser-100MB, SB-Condenser-300MB-Lite, SB-Condenser-100MB-Full, and SB-Condenser-3GB, respectively. However, the same training parameters were used in the parameter setup. Therefore, throughout the process of describing the implementation method, we only represent the **SB-Condenser-300MB-Lite** model in the core main section.

5.4.1 Setting parameters of training model

Table 4: Training parameters of Pretrain Condenser.

Model Name	Train Epochs	Total Batch Size
Phobert Large	8	32

In setting the parameters for Pretrain Condenser, we chose Epochs as 8 for training, and set the batch size for both the evaluation and training sets as 8. Additionally, we set a parameter to increase the batch size of the model



(a) Overview about f1 Scores

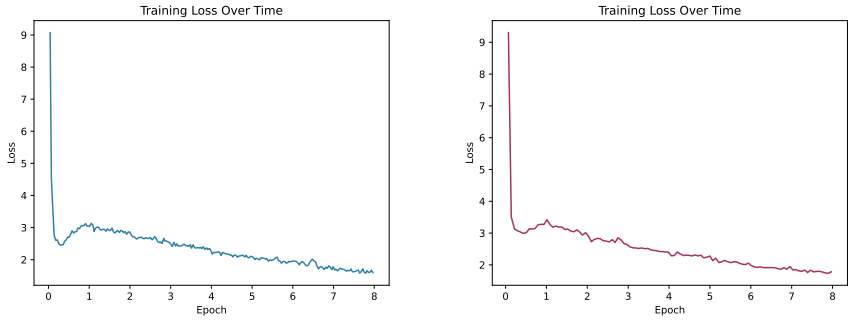
(b) Overview about cosim similarity 's accuracy, f1 and recall

Figure 12: Overview about SB-Condenser-300MB

to 32 by setting the Gradient Accumulation parameter to 4 (8*4). We used Gradient Accumulation to increase the batch size instead of directly increasing it to avoid exceeding the GPU memory limit. We used two types of GPUs with a memory limit of around 13GB, T4 from Google Colab and P100 from Kaggle Notebook. Directly increasing the batch size beyond a certain limit would have led to memory overflow problems. By using Gradient Accumulation, we ensured that the batch size could be increased up to a maximum of 32 without memory issues. It was important to avoid having a batch size that was too low as it would cause problems with the loss function during training.

5.4.2 Visualize Results

Looking at Figure 12, we have two charts to illustrate the variation of the Loss Function throughout the training process. We chose to visualize this parameter because it varies but always tends to decrease deeply from 7 to below 1. This proves that the training of our Condenser model is always efficient and the resulting output can ensure the quality for subsequent training cycles. Additionally, we can see that our training methodology for the two SB-Condenser-300MB-Lite and SB-Condenser-300MB-Full trials in this training process is completely different. With 100MB of original data from Zalo for the Lite version and 300MB with 200MB of data collected from reputable legal text websites such as "lawnet.vn" and "vbpl.vn" for the Full version. However, the training process shown in the two charts is quite similar, indicating that our preprocessing process works well and our data processing results are equivalent to the data in the "Legal Text Retrieval" dataset of the "Zalo Challenge 2021" competition.



(a) Overview about f1 Scores

(b) Overview about cosim similarity 's accuracy, f1 and recall

Figure 13: Overview about SB-Condenser-300MB

5.5 Pretrain Cocondenser

Describe presentation: We have a total of 4 experimental models, which are SB-Condenser-100MB, SB-Condenser-300MB-Lite, SB-Condenser-100MB-Full, and SB-Condenser-3GB, respectively. However, the same training parameters were used in the parameter setup. Therefore, throughout the process of describing the implementation method, we only represent the **SB-Condenser-300MB-Lite** model in the core main section.

5.5.1 Setting parameters of training model

Table 5: Training parameters of Pretrain Cocondenser.

Model Name	Train Epochs	Total Batch Size
Phobert Large	8	16

In setting the parameters for Pretrain Cocondenser, we chose Epochs as 8 for training, and set the batch size for both the evaluation and training sets as 16. In this section, gradient accumulation cannot be applied because using this parameter will result in very long training times when training on either a 100MB or 300MB training set with the Phobert Large language model.

5.5.2 Visualize Results

Looking at the Figure 13, we have two charts to illustrate the variation of the Loss Function parameter throughout the training process. The two charts show a similarity in shape and trend, with a decrease from 9 to around 2.5

and fluctuations in the range of 2.5 to 3. However, in the remaining part of the graph, a decreasing trend can be observed, with a decrease of nearly 1.5 in the last epoch. This also proves that the model works well, and the training process is continuous and effective, as the fluctuation of the loss function is not significant enough to occur in only one or two epochs, while the remaining part of the graph shows a decreasing trend.

5.6 Sentence Tranformer

5.6.1 Evaluation methods and indicators

Cosine similarity is a measure used to determine how similar two vectors are to each other. This measure is commonly used in text analysis and natural language processing applications to compare the similarity of two documents or pieces of text.

In a vector space, two vectors A and B can be represented by their respective components, where the value of each component represents the magnitude of that dimension of the vector. The cosine similarity between two vectors A and B is calculated as the cosine of the angle between the two vectors, which can be derived from the dot product of the two vectors and the product of their magnitudes.

The dot product of two vectors A and B is calculated by multiplying each corresponding component of the two vectors and adding the results. The magnitude of a vector can be calculated as the square root of the sum of the squares of its components. Using these formulas, we can compute the cosine similarity between two vectors A and B as follows:

$$cosine_{similarity}(A, B) = \frac{dot_{product}(A, B)}{magitude_A * magnitude_B} \tag{37}$$

The resulting cosine similarity value will range between -1 and 1, where -1 indicates that the two vectors are completely dissimilar, 0 indicates that the two vectors are orthogonal (i.e. perpendicular), and 1 indicates that the two vectors are identical.

Cosine similarity is a useful measure for comparing the similarity of two vectors in a high-dimensional space, where other distance measures such as Euclidean distance may not be as effective. It is also useful in applications such as collaborative filtering, where it can be used to recommend items to users based on their similarity to other users or items.

Manhattan distance is a measure used to calculate the distance between two points in a coordinate system. This measure is also known as taxicab distance, since it is analogous to the distance a taxi would have to travel on a grid-like city street system to get from one point to another.

In a two-dimensional coordinate system, Manhattan distance between two points P and Q is calculated as the sum of the absolute differences of their coordinates in the x- and y-dimensions, i.e.:

$$manhattan_{distance}(P,Q) = |P_x - Q_x| + |P_y - Q_y| \tag{38}$$

In higher-dimensional spaces, the Manhattan distance between two points can be computed by summing the absolute differences of their coordinates in each dimension.

Manhattan distance is often used in machine learning applications to calculate the distance between two data points in a feature space. It is particularly useful in cases where the features have different units or scales, since it is insensitive to changes in scale and orientation of the coordinate system.

One drawback of Manhattan distance is that it does not take into account the diagonal distance between two points, which may be a more relevant measure in certain applications. In such cases, Euclidean distance or Chebyshev distance may be more appropriate measures to use.

Euclidean distance is a commonly used measure of the distance between two points in a two- or multi-dimensional space. It is named after the ancient Greek mathematician Euclid, who first described this concept in his work on geometry.

In a two-dimensional space, Euclidean distance between two points P and Q can be calculated using the Pythagorean theorem, which states that the square of the hypotenuse of a right triangle is equal to the sum of the squares of the other two sides. Specifically, the Euclidean distance between two points P and Q is given by:

$$Euclidean_{distance}(P,Q) = sqrt((P_x - Q_x)^2 + (P_y - Q_y)^2) \tag{39}$$

In a multi-dimensional space, the formula for Euclidean distance is similar but involves summing the squares of the differences in each dimension and taking the square root of the sum:

$$Euclidean_{distance}(P,Q) = sqrt((P_1 - Q_1)^2 + (P_2 - Q_2)^2 + ... + (P_n - Q_n)^2) \tag{40}$$

Euclidean distance is a useful measure of distance in many applications, such as machine learning and data analysis. One advantage of Euclidean distance is that it takes into account both the x- and y-directions in a two-dimensional space, as well as all dimensions in a multi-dimensional space, which may be important in certain applications. However, it may be sensitive to differences in scale and orientation of the coordinate system, and may not be appropriate in all situations. In such cases, other distance measures such as Manhattan distance or Chebyshev distance may be more appropriate to use.

Accuracy: Accuracy is a measure of how often the model correctly predicts the label of a sample. It is calculated by dividing the number of correct predictions by the total number of predictions.

F1 score: F1 score is the harmonic mean of precision and recall, two other common evaluation metrics. It is a good metric to use when the classes are

imbalanced, meaning one class has significantly more samples than the other. F1 score ranges from 0 to 1, with higher values indicating better performance.

Recall: Recall, also known as sensitivity, is a measure of how well the model correctly identifies positive samples. It is calculated by dividing the number of true positives by the sum of true positives and false negatives.

Average Precision (AP): Average Precision is a metric commonly used in object detection and image segmentation tasks. It measures the area under the precision-recall curve, which shows the trade-off between precision and recall at different classification thresholds. AP ranges from 0 to 1, with higher values indicating better performance.

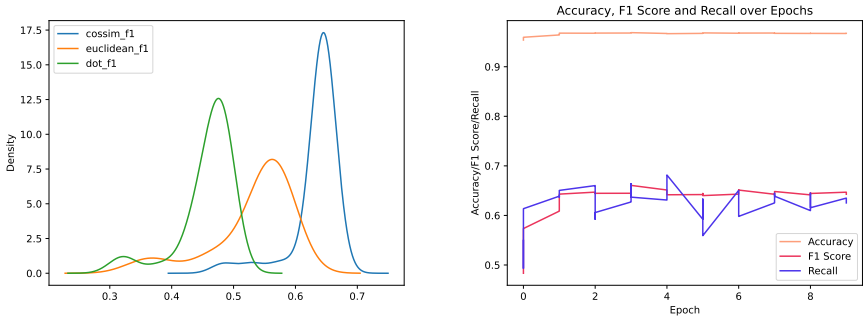
5.6.2 Setting parameters of training model

Table 6: Training parameters of Sentence Transformer.

Model Name	Train Epochs	Total Batch Size
Phobert Large(R1)	10	32
Phobert Large(R2)	5	32

In terms of Sentence Transformer, we divide into two round. For the first round, we set the training parameters as follows: max length is 256. This is because for the Phobert Base and Phobert Large language models, the maximum length per line is 256. Therefore, setting the max length to 256 ensures that each line meets the language model's requirements without causing errors. We also set the batch size parameter to 32 to ensure that the model's learning process runs optimally. We cannot increase the batch size to 64 or higher because our available resources are limited to the A30 with 24GB of memory. Increasing the batch size beyond this limit may cause memory overflow. While gradient accumulation can be used to increase the batch size without changing memory requirements, it may slow down the process. Therefore, this parameter is not supported in this case. We trained the model for 10 epochs because we found that it was suitable in terms of time while still ensuring the model's quality. Additionally, we consulted other papers as [4] with similar epoch numbers and achieved good results.

For the second round, we inherited the results from the last epoch checkpoint of the first round and continued to use the same training parameters, with a max length of 256 and a batch size of 32. However, this time we set the number of epochs to 5 because in some of our experiments and in other papers that use the Condenser architecture, the number of epochs ranges from 5-10. However, we found that using more than 5 epochs within this range did not result in significant changes. After round 2, we could continue training in the subsequent rounds, but training the model in these rounds only improves local results, while



(a) Overview about f1 Scores

(b) Overview about cosim similarity 's accuracy, f1 and recall

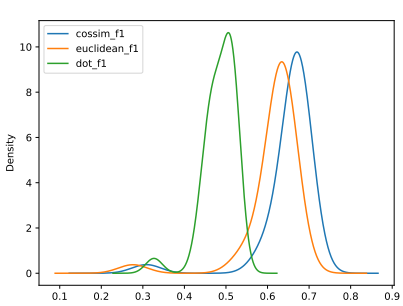
Figure 14: Overview about SB-Condenser-300MB-Full (Round 1)

overall there may be significant differences. To clarify this, we can observe the results we recorded in rounds 1 and 2, and from there, gain insight for subsequent rounds if we continue training.

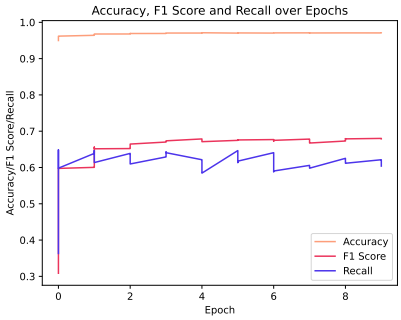
5.6.3 Visualize Results

With regard to Figure 14, Firstly, considering figure 14a, we can see that the "Overview about f1 Scores" chart, which evaluates three main methods: Cosine similarity, Manhattan distance, and Euclidean distance, shows that the variation of f1 scores during the training process is completely different. The variability of these three methods is completely different. The highest one is Cossim with a fluctuation range from 0.6 to 0.7, while the other two methods are much lower. Next is figure 14b, in which we focus on F1 Score, Recall, and Accuracy of the Cosine similarity evaluation method, as its displayed results are the best and its range represents more clearly the other evaluation methods. Throughout the training process, the fluctuation range of Accuracy is always above 0.97

In terms of Figure 15, Firstly, considering figure 15a, we can see that the "Overview about f1 Scores" chart, which evaluates three main methods: Cosine similarity, Manhattan distance, and Euclidean distance, shows that the variation of f1 scores during the training process is completely different. However, the density at the 0.7 level always has the highest proportion. Next is figure 15b, in which we focus on F1 Score, Recall, and Accuracy of the Cosine similarity evaluation method, as its displayed results are the best and its range represents more clearly the other evaluation methods. Throughout the training process, the fluctuation range of Accuracy is always above 0.95, which can be considered as a measure to ensure that the model performs relatively well.



(a) Overview about f1 Scores



(b) Overview about cosim similarity 's accuracy, f1 and recall

Figure 15: Overview about SB-Condenser-300MB-Lite (Round 1)

Table 7: Binary Accuracy Evaluation of Round 2

Evaluatio	Accuracy	F1	Precission
SB-Condenser-300MB-Lite			
Cosine-Similarity	96.9	63.2	76.3
Manhattan-Distance	96.6	59.2	69.4
Euclidean-Distance	96.6	59.5	73.8
SB-Condenser-300MB-Full			
Cosine-Similarity	96.9	63.8	72.7
Manhattan-Distance	96.4	54.3	68.1
Euclidean-Distance	96.4	54.5	67.5

As mentioned above, in the second round, we performed with 5 epochs and obtained relatively good results. With Accuracy always above 96.9 and F1 Score reaching approximately 63, and Precision can be close to 76.3 for SB-Condenser-300MB-Lite. Besides, in terms of SB-Condenser-300MB-Full having 96.9, 63.8 and 72.7 for accuracy, F1 Score and Precission in that order. It can be seen that the F1 Score in the second round is lower than in the first round, however, overall, other metrics achieved a better ratio than in the first round, such as Precision. That is why we stopped after the second round because if we continued to train in the following rounds, we would only improve locally on some specific types, while the overall performance would have large fluctuations, and the metrics could even decrease.

5.7 Final Results of Vietnamese Legal Text Retrieval

Table 8: The results of legal text retrieval versions

The metrics	Training data	F1 Score(val)
SB-Condenser-100MB	100MB	0.61
SB-Condenser-300MB-Lite	300MB	0.63
SB-Condenser-300MB-Full	300MB	0.63
SB-Condenser-3GB	3GB	0.66

The result: with regard to the summary table 8, we introduce four versions of a conference we tested based on the Condenser architecture to solve the problem of "Vietnamese Legal Text Retrieval".

- **In the first version (SB-Condenser-100MB):** we use 100MB of data from the Zalo Legal Text 2021 competition. We use 100 MB of data to pretrain Masked Language Model, although it is not much, it helps the language model understand the characteristics of words in the legal field linked together. In the following rounds, Condenser and Coconder, we continue to use this data to understand the context of each sentence and the links between legal terms and the separate content of each term.
- **In the second version (SB-Condenser-300MB-Lite):**, we added 200MB of data collected from reputable legal websites in Vietnam to supplement the initial 100MB data. First, we used the 300MB data to fine-tune the Phobert language model. Then, we used the checkpoint obtained from fine-tuning to train in subsequent rounds, with 100MB of data retained for training in each round.
- **In the third version (SB-Condenser-300MB-Full):**, we used 300MB of data similar to experiment 2, but this time we trained all rounds with the 300MB data, including Pretrain Masked Language Model, Pretrain Condenser, Pretrain Cocondenser. For the final round, Sentence Transformer, we reused 100MB because the task was to combine Sparse Retrieval and Dense Retrieval on the domain dataset to answer questions.
- **In the fourth version (SB-Condenser-3GB):**,we added 2.9 GB of data collected from reputable legal websites in Vietnam to supplement the initial 100MB data. First, we used the 3GB data to fine-tune the Phobert language model. Then, we used the checkpoint obtained from fine-tuning to train in subsequent rounds, with 100MB of data retained for training in each round. Besides, This training method is similar to the second version, which can bring optimal training time while still ensuring that the results are similar to using all 3GB for training in all rounds.

We trained a total of 4 versions on different amounts of data: 100MB, 300MB, and 3GB. In particular, for the 300MB data, we split it into two training methods. First, with the SB-Condenser-300MB-Lite version, we used 300MB of training data for pretraining the masked language model, while in subsequent iterations, we used the original 100MB data in the Zalo domain for the SB-Condenser-300MB-Full experiment, training with 300MB for all steps. The results of the two 300MB experiments exceeded 0.63, which opened up a new training approach where we can apply the training method of the SB-Condenser-300MB-Lite version to save time and training resources. For the 3GB experiment, we applied the training method of the 300MB-Lite version and achieved an F1 score of over 0.66.

Predicted class		
Actual class	Positive	Negative
Positive	**True Positive (TP)**	False Negative (FN)
Negative	False Positive (FP)	**True Negative (TN)**

Table 9: Confusion matrix for binary classification

F2 Score: being a metric that combines precision and recall. It is calculated as follows:

$$F_2 = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{41}$$

- where β is a parameter that adjusts the relative weight of precision and recall. In practice, F2 score is often used when recall is more important than precision. For example, in medical diagnosis, it is more important to correctly identify all positive cases (high recall) even if it means more false positives (low precision).
- To calculate F2 score, we first need to calculate precision and recall from the confusion matrix.

Precisio: being the ratio of true positives to the sum of true positives and false positives:

$$precision = \frac{TP}{TP + FP} \tag{42}$$

Recall: being the ratio of true positives to the sum of true positives and false negatives:

$$recall = \frac{TP}{TP + FN} \tag{43}$$

Table 10: The F2 Score of legal text retrieval versions

Version	Metric	Score
SB-Condenser-300MB-Lite	F2	0,699
SB-Condenser-300MB-Full	F2	0,649
SB-Condenser-3GB	F2	0.723

In terms of the table 10, we evaluated our trial version using the F2 Score evaluation method. In the field of law, Recall plays a crucial role as it indicates the proportion of predictions that match the labels. Our highest F2 Score result was achieved with the SB-Condenser-3GB version, with a score of 0.723. The Lite version trained on 300MB achieved a score of 0.699, while the Full version achieved a score of 0.649.

For inference, take a look at Figure 16, the query is sent to BM25+ module to calculate BM25+ score with all documents in the database (sparse retrieval purpose). At the same time, the query is passed to SentenceBERT model to create cosine similarity score with all documents in the database (dense retrieval purpose). The documents are chosen whose scores is maximum and calculated by $BM25 + \_score * cosine\_similarity$. Despite the number of documents returned is fixed but it has another constraint that the relevant documents need to lie in the range from $max\_score - 2.6$ to max_score. finally, the documents and query are given to Question Answering model to extract a keypoint that the query mentions, avoid redundant information in original returned documents from the retrieval process' output.

5.8 Final Results of Question Answering

The F1 Score being a commonly used metric for classification problems, particularly in QA. It is suitable when both precision and recall are equally important. To calculate the F1 score in this context, the individual words in the prediction and true answer are compared. The score is based on the number of words shared between the prediction and the truth. Precision is determined by dividing the number of shared words by the total number of words in the prediction, while recall is determined by dividing the number of shared words by the total number of words in the ground truth.

Exact Match: The measurement referred to is very straightforward. In each case where a question and corresponding answer are given, if the model's prediction matches the characters of the true answer(s), the EM score is 1. However, any deviation in character accuracy will result in an EM score of 0. This is a binary measure, meaning that being even one character off results in a score of 0. Furthermore, if the model predicts any text for a negative example,

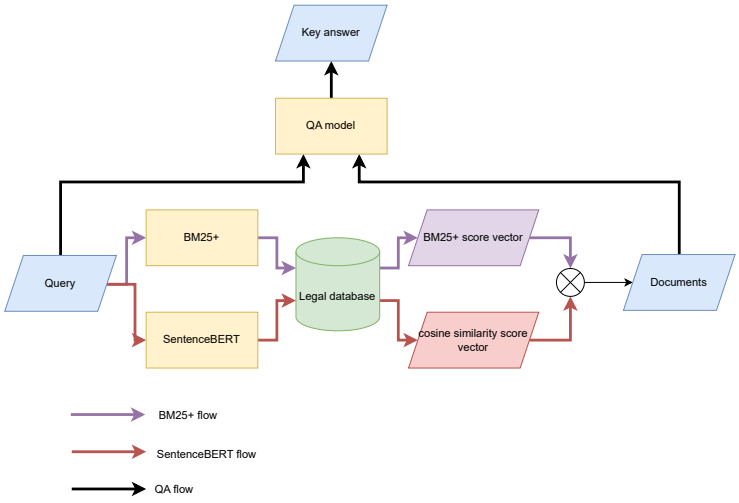


Figure 16: Inference flow

it automatically receives a score of 0 for that particular example.

$$F1 = \frac{(1+\beta^2) * precision * recall}{\beta^2 * precision + recall} \tag{44}$$

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) being a set of metrics used for automatic evaluation of text summarization and machine translation. It measures the similarity of the generated text with one or more reference summaries or translations, using several scores such as ROUGE-1, ROUGE-2, and ROUGE-L. These scores represent the n-gram overlap of the generated text with the reference text at the unigram, bigram, and longest common subsequence levels. ROUGE metrics are widely used in natural language processing and text summarization research to evaluate the quality of generated summaries or translations.

Table 11: The result of question answering version (Vqa-ViT5)

Version	Metric	Score
Vqa-ViT5	F1	0,646
Vqa-ViT5	EM	0,41
Vqa-ViT5	rougeL	0,66

Dataset: In order to address the problem of limited training data for legal question answering, the authors trained their legal Phobert model with UIT-ViQuAD [39], a collection of 23,000 questions and answers created by humans using passages from 174 Vietnamese Wikipedia entries. By doing this, the extractive question answering model can first learn reading comprehension skills before being applied or performed inference on the 520 legal questions and 1377 articles from the Automated Legal Question Answering Competition (ALQAC 2022). After checking 520 questions, we found that there are 9 questions contain the answer is not a pieces of information that can be extracted from a given question's corresponding articles. Because we trained our Vqa-ViT5 to perform extractive question answering task, we discard these unsuitable question and retain 511 samples

The results: We used three evaluation methods, namely F1 Score, Exact Match, and ROUGE, to evaluate the performance of our Vqa-ViT5 experiment, which was trained on the dataset UIT-ViQuAD [39] with data spanning various domains. For evaluation, we used 511 pairs of legal questions and answers from the ALQAC-2022 competition, in contrast to the commonly used approach of training on legal data and using a small set of validation data to evaluate results. Table 11 shows our Vqa-ViT5 model trained on the comprehensive dataset and using all 511 questions achieved an F1 Score of 0.646. The more accuracy-demanding evaluation method, Exact Match (EM), yielded a score of 0.41. [3] gives their result about 90% on ALQAC-2022. However, they mentions that their F1 score is on the dev set which they randomly pick 15% of the official dataset. It means their validation dataset (about 78 data) is smaller than us (511 data). Moreover, our Vqa-ViT5 is not trained on ALQAC-2022 dataset at all but it still gives acceptable result. Table 12 shows some examples from Vqa-ViT5, at the first question, model predict correctly answer in token-level, obviously its EM is 1.0. Take a look at the second question, our model predicts a span of text that includes the actual label, although the prediction is not incorrect, but EM metric still return 0.0, and most of inference result is the same with this situation, that why we said the result is kind of acceptable but the EM score is not pretty high. Therefore In Table 12, we also compute rougL score in order not to discard acceptable result like EM score because rougL score measures the similarity between a machine-generated summary or translation and a reference summary or translation by computing the longest common subsequence (LCS) between them. The LCS is the longest sequence of words that appears in both the generated and reference summaries. However, result of rougeL score is just approximate to indicate that there are some semantically correct predicted answers are discarded by EM, rougeL metric is not popularly used for extractive question answering task because of answer's representation.

Table 12: Vqa-ViT5 example result table

question:	Chiếm đoạt di vật của tử sĩ có thể bị phạt tù lên đến bao nhiêu năm? (Appropriating relics of martyrs can be punished with up to how many years?)
context:	Tội chiếm đoạt hoặc hủy hoại di vật của tử sỹ ...thì bị phạt tù từ 02 năm đến **07 năm**: a) Là chỉ huy hoặc sĩ quan; b) Chiếm đoạt hoặc hủy hoại di vật của 02 tử sỹ trở lên. (The crime of appropriating or destroying the relics of martyrs ... shall be punishable by imprisonment from 02 years to **07 years**: a) Being a commander or officer; b) Appropriating or destroying relics of 02 or more martyrs.)
prediction:	07 năm (07 years)
label:	07 năm (07 years)
question:	Người đã nhận làm gián điệp, nhưng không thực hiện nhiệm vụ được giao và tự thú, thành khẩn khai báo với cơ quan nhà nước có thẩm quyền, thì được miễn trách nhiệm gì về tội gián điệp? (A person who has accepted to act as a spy, but fails to perform the assigned tasks and confesses and honestly declares to the competent state agency, shall be exempt from any responsibility for espionage charges?)
context:	Tội gián điệp 1. Người nào có một trong các hành vi... a) Hoạt động tình báo, ... thành khẩn khai báo với cơ quan nhà nước có thẩm quyền, thì được miễn trách nhiệm **hình sự** về tội này. (Crime of espionage 1. Any person who commits one of the acts... a) Intelligence activities, ... sincerely declares to the competent state agency, shall be exempt from responsibility **criminal** about this crime.)
prediction:	miễn trách nhiệm hình sự (exempt from criminal liability)
label:	hình sự (Criminal)

6 DISCUSSIONS

The general topic of our thesis is "Vietnamese Legal Text Retrieval," where our model aims to provide users with the most relevant and appropriate laws in response to their query. However, to improve the answering capability of our model, we have also developed a "Question Answering" task that extracts laws related to the user's query and presents a concise answer based on those laws. This is to provide users with the most accurate and succinct answer possible, tailored to their specific question.

The results of our work over the past 14 weeks will be presented at the ISICO 2023 conference in July in Indonesia. Besides, we will present the three main parts of our work, which are also documented in two papers: preprocessing of Vietnamese legal texts, which we collected from two websites, "lawnet.vn" and "vbpl.vn," with nearly 145,000 documents; training the Condenser architecture to achieve optimal results on large datasets; and using the ViT5 model trained on multidisciplinary datasets to achieve top results when tested on legal datasets.

Vietnamese Legal Text Retrieval: we conducted experiments on a total of 4 trial versions, namely SB-Condenser-100MB, SB-Condenser-300MB-Lite, SB-Condenser-300MB-Full, and SB-Condenser-3GB. We compared the training approach between the 300MB-Lite and 300MB-Full versions, which proposed training on larger datasets like 3GB to save costs and reduce total training time. This is because after evaluating the learning process or results through several training iterations, they produced similar results. Therefore, this can be considered a relatively new approach that brings efficiency for cases with limited time and training resources.

Question Answering: we conducted training on the UIT-ViQuAD dataset with nearly 23,000 question-answer pairs spanning various domains in Vietnamese. After training, we evaluated on 510 question-answer pairs from the ALQAC 2022 competition and achieved results comparable to those of other published works that mainly focus on legal data. This opens up a new approach for specific topics such as law because the data for training "Question Answering" models is currently scarce in most languages. Typically, to increase the accuracy of the model, data is outsourced to a data expert to prepare for the training process. However, for legal data, it is necessary to involve legal experts such as legal specialists or lawyers to evaluate the dataset used to train the model from a third party or directly from those who participate in completing it. This approach can be applied not only to Vietnamese but also to other languages and different topics since finding a multi-domain dataset is more common and easier than complex topics like law.

7 CONCLUSIONS

Along with the experiments on my approaches, architectures, and proposals that I have built and evaluated over the past 14 weeks, we have achieved certain results on two tasks, "Vietnamese Legal Text Retrieval" and "Question Answering," equivalent to the top results on these tasks in Vietnam in 2022. From that, we have gained a better understanding of the opportunities and challenges in developing projects in practice, such as the need to quickly and timely collect and process legal data with relative high costs and time required to train complete models for both tasks. However, this topic still holds a lot of potential for practical applications in addressing small issues from individuals' legal research to protect their legitimate rights or even for foreign businesses wishing to invest in Vietnam but face legal obstacles as it is now.

In the future, we will continue to develop this project with more accurate data processing, as well as integrating new algorithms to further enhance the effectiveness. Perhaps releasing an official version for user experience evaluation will help us collect more accurate evaluation results and provide us with a more comprehensive overview of the applicability of this topic in Vietnam today. In addition, we will also consider research directions to ensure that the results of the model are improved and continue to produce papers at conferences on Natural Language Processing in the future.

8 REFERENCES

References

[1] Luyu Gao and Jamie Callan. Condenser: a pre-training architecture for dense retrieval. *arXiv preprint arXiv:2104.08253*, 2021.

[2] Dat Quoc Nguyen and Anh Tuan Nguyen. Phobert: Pre-trained language models for vietnamese. *arXiv preprint arXiv:2003.00744*, 2020.

[3] Hieu Nguyen Van, Dat Nguyen, Phuong Minh Nguyen, and Minh Le Nguyen. Miko team: Deep learning approach for legal question answering in alqac 2022. In *2022 14th International Conference on Knowledge and Systems Engineering (KSE)*, pages 1–5. IEEE, 2022.

[4] Nhat-Minh Pham, Ha-Thanh Nguyen, and Trong-Hop Do. Multi-stage information retrieval for vietnamese legal texts. *arXiv preprint arXiv:2209.14494*, 2022.

[5] Phi Manh Kien, Ha-Thanh Nguyen, Ngo Xuan Bach, Vu Tran, Minh Le Nguyen, and Tu Minh Phuong. Answering legal questions by learning neural attentive text representation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 988–998, 2020.

[6] Chieu-Nguyen Chau, Truong-Son Nguyen, and Le-Minh Nguyen. Vnlawbert: A vietnamese legal answer selection approach using bert language model. In *2020 7th NAFOSTED Conference on Information and Computer Science (NICS)*, pages 298–301, 2020.

[7] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[11] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[12] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[13] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[14] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[15] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49, 2004.

[16] Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*, pages 232–241. Springer, 1994.

[17] Qiao Jin, Andrew Shin, and Zhiyong Lu. Lader: Log-augmented dense retrieval for biomedical literature search, 2023.

[18] Avirup Sil, Jaydeep Sen, Bhavani Iyer, Martin Franz, Kshitij Fadnis, Mihaela Bornea, Sara Rosenthal, Scott McCarley, Rong Zhang, Vishwajeet Kumar, Yulong Li, Md Arafat Sultan, Riyaz Bhat, Radu Florian, and Salim Roukos. Primeqa: The prime repository for state-of-the-art multilingual question answering research and development, 2023.

[19] Qiuhong Zhai, Wenhao Zhu, Xiaoyu Zhang, and Chenyun Liu. Contrastive refinement for dense retrieval inference in the open-domain question answering task. *Future Internet*, 15(4):137, 2023.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[21] Ruiyang Ren, Shangwen Lv, Yingqi Qu, Jing Liu, Wayne Xin Zhao, QiaoQiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. PAIR: Leveraging passage-centric similarity relation for improving dense passage retrieval. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Association for Computational Linguistics, 2021.

[22] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2010.08191*, 2020.

[23] Nicholas Monath, Manzil Zaheer, Kelsey Allen, and Andrew McCallum. Improving dual-encoder training through dynamic indexes for negative mining, 2023.

[24] Xuan Fu, Jiangnan Du, Hai-Tao Zheng, Jianfeng Li, Cuiqin Hou, Qiyu Zhou, and Hong-Gee Kim. Ss-bert: A semantic information selecting approach for open-domain question answering. *Electronics*, 12(7):1692, 2023.

[25] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model?, 2020.

[26] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering, 2021.

[27] Peng Xu, Davis Liang, Zhiheng Huang, and Bing Xiang. Attention-guided generative models for extractive question answering, 2021.

[28] Thanh Vu, Dat Quoc Nguyen, Dai Quoc Nguyen, Mark Dras, and Mark Johnson. VnCoreNLP: A Vietnamese natural language processing toolkit. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 56–60, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[29] Dat Quoc Nguyen, Dai Quoc Nguyen, Thanh Vu, Mark Dras, and Mark Johnson. A fast and accurate vietnamese word segmenter. In Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Kôiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA), 2018.

[30] Dat Quoc Nguyen, Thanh Vu, Dai Quoc Nguyen, Mark Dras, and Mark Johnson. From word segmentation to POS tagging for Vietnamese. In *Proceedings of the Australasian Language Technology Association Workshop 2017*, pages 108–113, Brisbane, Australia, December 2017.

[31] Feng Wang and Huaping Liu. Understanding the behaviour of contrastive loss. *CoRR*, abs/2012.09740, 2020.

[32] Nhat-Minh Pham, Ha-Thanh Nguyen, and Trong-Hop Do. Multi-stage information retrieval for vietnamese legal texts, 2022.

[33] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does bert look at? an analysis of bert's attention, 2019.

[34] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering, 2021.

[35] Luyu Gao and Jamie Callan. Unsupervised corpus aware language model pre-training for dense passage retrieval. *arXiv preprint arXiv:2108.05540*, 2021.

[36] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[37] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[38] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval, 2020.

[39] Kiet Van Nguyen, Duc-Vu Nguyen, Anh Gia-Tuan Nguyen, and Ngan Luu-Thuy Nguyen. A vietnamese dataset for evaluating machine reading comprehension, 2020.

9 APPENDIX

Appendix. Source code & Dataset

	Link
Source code	https://drive.google.com/drive/folders/13MKB2i29prZ8KN-Kv5dNsnvb8v2QpBEO?usp=sharing
Dataset	https://drive.google.com/drive/folders/1i08yDyb_Z-BoppN3VMk7Tham1rs_rlqe?usp=sharing