**Vietnam General Confederation of Labor**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**

****

**FINAL PROJECT**

**Natural Language Processing**

**Understanding, Fine-tuning, and Evaluating Transformer Models on Vietnamese Data**

*Instructor*: Le Anh Cuong

*Student*: Le Quang Huy– 521H0238

Tactay Hoang Jon – 522H0155

HO CHI MINH CITY, 2025

**Vietnam General Confederation of Labor**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**

****

**FINAL PROJECT**

**Natural Language Processing**

**Understanding, Fine-tuning, and Evaluating Transformer Models on Vietnamese Data**

*Instructor*: Le Anh Cuong

*Student*: Le Quang Huy– 521H0238

Tactay Hoang Jon – 522H0155

HO CHI MINH CITY, 2025

**ACKNOWLEDGEMENT**

The first sincere thanks I want to give to Mr. Le Anh Cuong , who enthusiastically taught and worked tirelessly to give me enough tools and skills to complete this report. He played an important role in improving my mathematical logic and knowledge. The second thanks I would like to give to the teachers of the Department of Information Technology of Ton Duc Thang University for giving me the opportunity to do this report.

Because the impact of the epidemic is too great, my report will have some errors, I am very open to receiving feedback from teachers so that I can improve my report writing skills.

Finally, I wish you good health and success in your noble career.

*Ho Chi Minh city, 1st December, 2025*

*Le Quang Huy*

*Tactay Hoang Jon*

**THIS PROJECT WAS COMPLETED AT**

**TON DUC THANG UNIVERSITY**

I fully declare that this is my own project and is guided by Mr.Le Anh Cuong ; The research contents and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

Besides that, the project also uses a number of comments, assessments as well as data from other authors, other agencies and organizations, with citations and source annotations.

Should any frauds were found, I will take full responsibility for the content of my report. Ton Duc Thang University is not related to copyright and copyright violations caused by me during the implementation process (if any).

*Ho Chi Minh city, 1st December, 2025*

*Le Quang Huy*

*Tactay Hoang Jon*

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Ho Chi Minh 1st December, 2025*

*(Sign and write full name)*

Evaluation section for grading instructor

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Ho Chi Minh 1st December, 2025*

*(Sign and write full name)*

INDEX

[Architecture and Attention Mechanism 6](#_Toc9300)

[Encoder only Models: 6](#_Toc24194)

[Encoder Decoder Models: 7](#_Toc18344)

[Encoder-Decoder Model Architecture: 7](#_Toc5333)

[Working of Encoder Decoder Model 8](#_Toc25697)

[Decoder only model: 10](#_Toc7609)

[Overview: 11](#_Toc21038)

[1.1 Library 12](#_Toc18949)

[1.2 Link 13](#_Toc16077)

[1.3 Data 13](#_Toc14270)

[1.3.1 Data Introduction 13](#_Toc17026)

[1.3.2 Data Origins and Characteristics 13](#_Toc3871)

[1.3.3 Labeling Schema 14](#_Toc14716)

[1.3.4 Statistics 14](#_Toc17656)

[1.3.5 Applications and Research Value 14](#_Toc4417)

[1.4 Preprocessing Methodology 15](#_Toc29915)

[1.5 Hyperparameters & Methods 17](#_Toc29334)

[1.5.1 Hyperparameters 17](#_Toc14844)

[1.5.2 Methods 17](#_Toc13177)

[1.6 Setting 17](#_Toc25508)

[1.6.1 Architecture 18](#_Toc6702)

[1.6.2 Hyperparameters 19](#_Toc2726)

[1.6.3 Performance Optimization 19](#_Toc362)

[1.6.4 Model Selection Strategy 19](#_Toc30536)

[1.7 Synthetic data 20](#_Toc12065)

[1.8 Audit & Merge data 21](#_Toc873)

[1.9 Compare 22](#_Toc31369)

[1.9.1 Baseline data & Augmented data 22](#_Toc17139)

[1.9.2 F1-score 23](#_Toc833)

[1.9.3 Accuracy 23](#_Toc19683)

[1. Transformer architecture and the T5 “text-to-text” idea 24](#_Toc2763)

[2. ViT5 – a T5-style model for Vietnamese 24](#_Toc4446)

[3. Abstractive summarization 24](#_Toc19123)

[4. Fine-tuning a pre-trained encoder–decoder 25](#_Toc19892)

[5. Traditional automatic metrics: ROUGE, BLEU, BERTScore 25](#_Toc13599)

[6. Synthetic data augmentation with LLMs 26](#_Toc30020)

[1. Autoregressive (decoder-only) language models 27](#_Toc14998)

[2. The Qwen and Qwen3 27](#_Toc24209)

[3. Perplexity as a traditional metric 27](#_Toc18236)

[4. LLM-as-a-Judge for generation quality 28](#_Toc15894)

[1. OpenHust Vietnamese Summarization Dataset 29](#_Toc31461)

[2. Vietnews – Vietnamese News Summarization Dataset 31](#_Toc1172)

[ViT5: Text Summerization 33](#_Toc31390)

[4.1. Overall Experimental Pipeline 33](#_Toc9455)

[4.2. Dataset Loading and Splitting 33](#_Toc28575)

[4.3. Data Cleaning and Filtering 34](#_Toc9686)

[4.3.1. Text cleaning / normalization (whitespace handling) 34](#_Toc9931)

[4.3.2. Length-based filtering (minimum character thresholds) 35](#_Toc17866)

[4.3.3. Scope of application 35](#_Toc7461)

[4.4. Tokenization and Sequence-to-Sequence Formatting 35](#_Toc27712)

[4.4.1. Input and target length constraints 35](#_Toc26843)

[4.4.2. Preprocessing function for summarization 35](#_Toc14579)

[4.5. Model and Training Objective 36](#_Toc11235)

[4.5.1. Base model selection 36](#_Toc19102)

[4.5.2. Optimization objective 36](#_Toc25655)

[4.6. Baseline Fine-tuning Configuration (Original Data Only) 36](#_Toc24763)

[4.6.1. Data collation for Seq2Seq training 36](#_Toc20274)

[4.6.2. Training hyperparameters 36](#_Toc22517)

[4.7. Inference and Decoding 38](#_Toc161)

[4.8. Traditional Automatic Evaluation 38](#_Toc16240)

[4.8.1. Metrics used 38](#_Toc4989)

[4.8.2. Metric computation procedure 38](#_Toc29975)

[4.8.3. Visualization of evaluation results 38](#_Toc18805)

[4.9. LLM-based Evaluation (Gemini as an Automatic Judge) 39](#_Toc27020)

[4.9.1. Summary generation for LLM evaluation 39](#_Toc22435)

[4.9.2. Scoring rubric and output format 39](#_Toc28788)

[4.9.3. Parsing and aggregation 40](#_Toc26669)

[4.10. Synthetic Data Generation (Paraphrased Summaries) 40](#_Toc30109)

[4.10.1. Sampling strategy 40](#_Toc7219)

[4.10.2. Paraphrase prompt constraints 40](#_Toc17451)

[4.11. Synthetic Data Quality Evaluation 40](#_Toc8133)

[4.12. Augmented Fine-tuning with Synthetic Data 41](#_Toc24161)

[4.12.1. Dataset construction and tokenization 41](#_Toc2690)

[4.12.2. Training set augmentation 41](#_Toc31156)

[4.12.3. Continued fine-tuning setup 41](#_Toc3859)

[4.13. Comparative Evaluation (Baseline vs. +Synthetic) 41](#_Toc11526)

[4.13.1. Consistent evaluation protocol 41](#_Toc25642)

[4.13.2. Result reporting and visualization 41](#_Toc19000)

[Qwen3: Text Generation 42](#_Toc31314)

[4.1. Overall Experimental Setup 42](#_Toc29992)

[4.2. Dataset Loading and Splitting 43](#_Toc3430)

[4.3. Data Cleaning and Filtering 43](#_Toc27608)

[4.3.1. Text cleaning / normalization (whitespace handling) 43](#_Toc22192)

[4.3.2. Length-based filtering (minimum character thresholds) 43](#_Toc20622)

[4.3.3. Scope of application 43](#_Toc3317)

[4.4. Tokenization and Input Formatting for Decoder-only Training 43](#_Toc3626)

[4.4.1. Sequence construction and length constraints 43](#_Toc2561)

[4.4.2. Prompt-based preprocessing and label masking 44](#_Toc28682)

[4.5. Model and Training Objective 44](#_Toc26990)

[4.5.1. Base model selection 44](#_Toc4662)

[4.5.2. Optimization objective 44](#_Toc21173)

[4.6. Baseline Fine-tuning Configuration (Original Data Only) 44](#_Toc13217)

[4.6.1. Data collation 44](#_Toc28884)

[4.6.2. Training hyperparameters 44](#_Toc965)

[4.7. Inference and Decoding Strategy 46](#_Toc27655)

[4.8. Automatic Evaluation Metrics 46](#_Toc3564)

[4.9. LLM-based Evaluation 46](#_Toc7786)

[4.10. Synthetic Data Generation 47](#_Toc14300)

[4.10.1. Sampling strategy 47](#_Toc17016)

[4.10.2. Paraphrase prompt constraints 48](#_Toc14014)

[4.11. Synthetic Data Quality Evaluation (Qwen3) 48](#_Toc4808)

[4.12. Augmented Fine-tuning with Synthetic Data (Qwen3) 48](#_Toc13443)

[4.12.1. Dataset construction and tokenization 48](#_Toc19580)

[4.12.2. Training set augmentation 48](#_Toc28936)

[4.12.3. Continued fine-tuning setup 48](#_Toc21039)

[4.13. Comparative Evaluation (Baseline vs. +Synthetic) – Qwen3 49](#_Toc23411)

[4.13.1. Consistent evaluation protocol 49](#_Toc26617)

[4.13.2. Result reporting and visualization 49](#_Toc15642)

OVERVIEW

This final project focuses on understanding, fine-tuning, and evaluating Transformer-based models on Vietnamese data. Students are expected to grasp the Transformer architecture and the three main model families (encoder-only, encoder–decoder, and decoder-only), fine-tune pretrained language models on a Vietnamese dataset they collect and preprocess, apply data augmentation using synthetic data generated by larger LLMs, and evaluate model performance with both traditional metrics (e.g., ROUGE, BLEU, perplexity, BERTScore) and LLM-based evaluation. Through this, students practice the full NLP pipeline—from modeling and training to critical analysis of results and discussion of generalization, strengths, and limitations of Transformer models for Vietnamese language tasks.

INTRODUCTION

Transformers have reshaped modern natural language processing by providing a unified, highly scalable architecture that can be adapted to many different tasks. Instead of designing a new model for each problem, practitioners now typically start from a large pretrained Transformer and fine-tune it on their target dataset. Within this ecosystem, three main families of Transformer models have become especially important: **encoder-only models** (such as RoBERTa and PhoBERT), which encode an input sequence into contextual representations and are particularly strong for classification and sequence-labeling; **encoder–decoder models** (such as T5 and ViT5), which map one text sequence into another and are naturally suited for conditional generation tasks like summarization, translation, or question answering; and **decoder-only models** (such as Qwen3 and GPT-style LLMs), which are trained to predict the next token and excel at open-ended text generation and dialogue.

This project uses these three families as a lens to systematically study Transformer models on **Vietnamese** data. For encoder-only models, we focus on fine-tuning RoBERTa-style Vietnamese models (e.g., PhoBERT) for supervised tasks such as text classification or token-level labeling, analysing how contextual representations help capture Vietnamese morphology and syntax. For encoder–decoder models, we fine-tune ViT5/T5 on conditional generation tasks where there is a clear input and output, with abstractive summarization in Vietnamese as a central case study. For decoder-only models, we fine-tune a compact Qwen3-0.6B model for free-form text generation and simple dialogue, exploring how small open-weight LLMs can be specialized to a particular Vietnamese domain.

Across these settings, the project also investigates two cross-cutting themes: **synthetic data augmentation** and **evaluation methodology**. We experiment with generating synthetic training examples using stronger LLMs, for example by paraphrasing summaries or creating new prompt–response pairs, and then studying how this additional data affects downstream performance. On the evaluation side, we combine traditional automatic metrics (Accuracy, F1, ROUGE, BLEU, Perplexity, BERTScore, etc.) with **LLM-based evaluation**, where a more powerful language model acts as a judge to rate the coherence, relevance, fluency and factuality of model outputs. By the end of the project, we aim not only to obtain working Vietnamese NLP systems, but also to draw broader conclusions about the strengths and weaknesses of encoder-only, encoder–decoder, and decoder-only Transformers, and about when synthetic data and LLM-based evaluation are most useful in practice.

# Architecture and Attention Mechanism

The Transformer architecture, introduced by Vaswani et al. in the paper *"Attention Is All You Need"* (2017), represents a paradigm shift in sequence modeling. Unlike Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks, the Transformer eschews recurrence and convolution entirely. Instead, it relies solely on attention mechanisms to draw global dependencies between input and output.

This section details the macroscopic architecture and the microscopic mathematical operations of the Attention mechanism.

# Encoder only Models:

An encoder-only architecture is a type of neural network, typically based on the transformer model, that focuses solely on processing and understanding an entire input sequence to create a rich, contextual representation. This architecture is designed for tasks that require deep analysis of the input rather than generating new output sequences.



Overview:

The encoder-only Transformer processes the entire input sequence simultaneously without recurrence or autoregressive generation. All tokens in the input are available at once, enabling parallel computation and bidirectional context modeling.

Unlike decoder-only architectures that generate tokens sequentially, the encoder-only Transformer focuses on learning contextual representations of the input. The output is a sequence of context-aware embeddings, one for each input token, which are used for understanding-oriented tasks rather than text generation.

Each encoder block consists of two main sublayers: a multi-head self-attention mechanism and a position-wise feed-forward network. Residual connections and layer normalization are applied after each sublayer to stabilize training and support deep architectures.

Multi-head self-attention allows every token to attend to all other tokens in the input sequence, capturing global dependencies in a bidirectional manner. This design enables encoder-only models to produce rich semantic representations that incorporate information from both left and right contexts.

The feed-forward network is applied independently to each token representation and introduces non-linearity, enhancing the expressive power of the model. Multiple identical encoder blocks are stacked sequentially, with each layer refining the representations produced by the previous one.

The final output of the encoder stack is a set of contextualized token embeddings. Depending on the downstream task, these embeddings can be pooled for sentence-level classification or used directly for token-level tasks such as named entity recognition

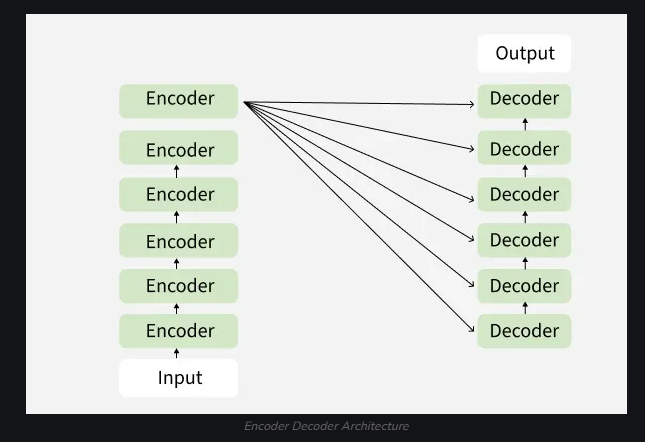
# Encoder Decoder Models:

Encoder-decoder model is a type of neural network that is mainly used for tasks where both the input and output are sequences. This architecture is used when the input and output sequences are not the same length for example translating a sentence from one language to another, summarizing a paragraph, describing an image with a caption or convert speech into text. It works in two stages:

* Encoder: The encoder takes the input data like a sentence and processes each word one by one then creates a single, fixed-size summary of the entire input called a context vector or latent space.
* Decoder: The decoder takes the context vector and begins to produce the output one step at a time.

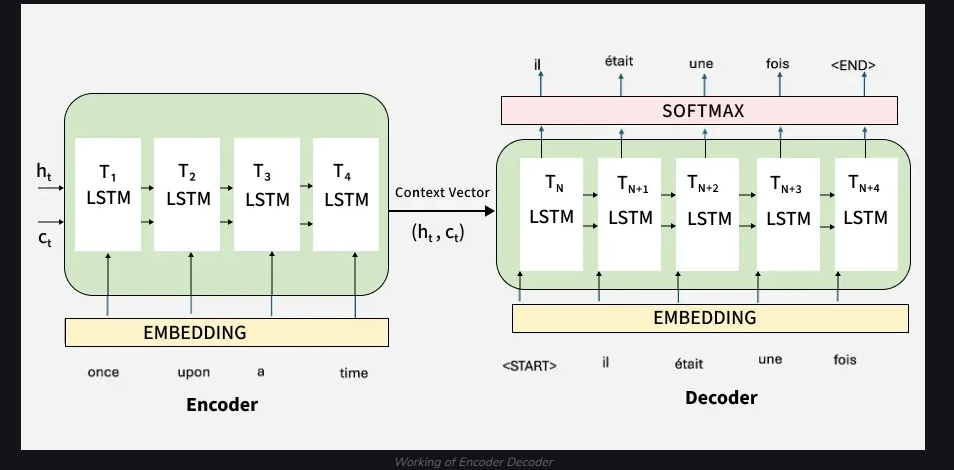
## **Encoder-Decoder Model Architecture:**

In an encoder-decoder model both the encoder and decoder are separate networks each one has its own specific task. These networks can be different types such as Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), Gated Recurrent Units (GRUs), Convolutional Neural Networks (CNNs) or even more advanced models like Transformers.



## Working of Encoder Decoder Model

The actual working of the encoder decoder model is shown in below diagram. Now we will understand it stepwise:



Step 1: Tokenizing the Input Sentence

The sentence "I am learning AI" is first broken into tokens: ["I", "am", "learning", "AI"].

Each word (token) is converted into a vector that a machine can understand. This process is called embedding.

Step 2: Encoding the Input

The Encoder processes these embeddings using self-attention.

Self-attention helps the encoder to focus on important words. For example while encoding "learning", it understands its relation with "I" and "AI."

After processing the encoder generates a Context Vector which captures the meaning of the entire sentence. For example in the image The arrows show how each word relates to the others during encoding. The final output from the encoder is the context representation

Step 3: Passing the Context to the Decoder

The Context Vector is passed to the Decoder as shown in image.

It acts like a summary of the full input sentence.

Step 4: Decoder Generates Output Step-by-Step

The Decoder uses the context and starts creating the output one word at a time.

First it predicts the first word then uses that to predict the second word and so on

Step 5: Decoder Attention

While generating each word the decoder attends to different parts of the input sentence to make better predictions.

For example when translating "learning," it might pay more attention to the word "learning" in the input.

Step 6: Producing the Final Output

The decoder continues generating until the full translated sentence is produced.

Each output token depends on the previous ones and the input context. You finally see the output tokens generated on the right side of the diagram completing the translation.

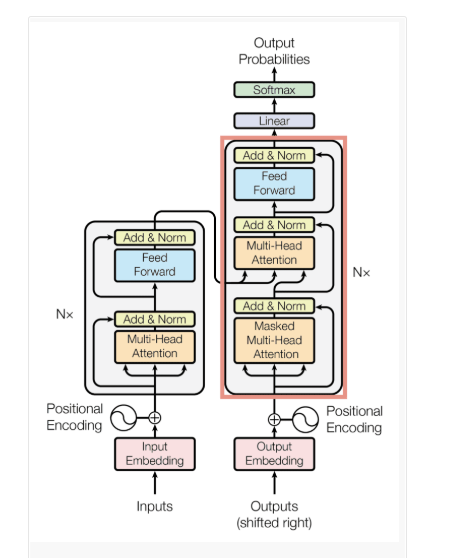
# Decoder only model:

Decoder-only Transformer is a specific type of neural network architecture that serves as the foundation for most modern Large Language Models (LLMs) like GPT. This architecture is primarily designed for autoregressive tasks, where the goal is to generate data sequentially, such as writing an email, completing code, or having a conversation. Unlike the Encoder-Decoder model which separates the understanding and generating phases, the Decoder-only architecture focuses purely on predicting the next element in a sequence based on the history of previous elements. It operates through two main mechanisms:

Masked Self-Attention Blocks: The core processing units take the input sequence and apply a "masked" attention mechanism. This allows the model to look back and attend to all previous words to understand the context, but it strictly blocks (masks) any access to future words. This ensures that the model cannot "see" the answer it is supposed to predict, forcing it to learn the causal relationship between words based solely on past information.

Next Token Prediction: After passing through the blocks, the processed data goes through a Linear layer and a Softmax function. This stage calculates the probability of every word in the vocabulary being the next one in the sequence. The model selects the most likely word (token), appends it to the existing sequence, and then feeds this new sequence back into the start to generate the subsequent word.

The following visualization gives an overview of the transformer architecture.



## Overview:

The input and output of a transformer:

The input is a prompt (often referred to as context) fed into the transformer as a whole. There is no recurrence.

The output depends on the goal of the model. For GPT models, the output is a probability distribution of the next token/word that comes after the prompt. It outputs one prediction for the complete input.

Key components:

The embedding: the input of the transformer model is a prompt. This prompt needs to be embedded into something that the model can use.

The block(s): This is the main source of complexity. Each block contains a masked multi-head attention submodule, a feedforward network, and several layer normalization operations. *Blocks are put in sequence* to make the model deeper.

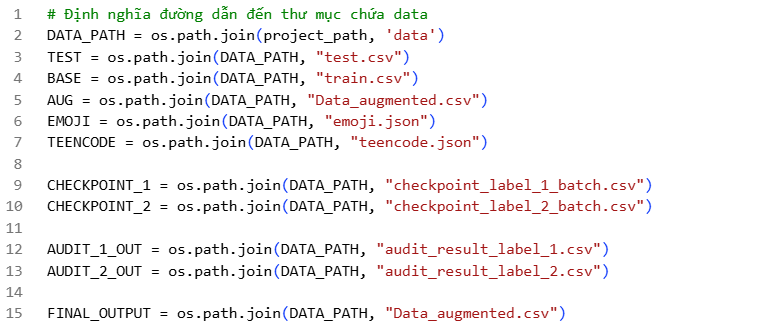
The output: the output of the last block is fed through one more linear layer to obtain the final output of the model (a classification, a next word/token etc.)

Task 1 – Text Classification with Bert

## **1.1 Library**



# 1.2 Link



## 

# 1.3 Data

## 1.3.1 Data Introduction

**ViHSD** is a large-scale benchmark dataset designed for the task of hate speech and offensive language detection in Vietnamese. This dataset was constructed and released by a research team affiliated with the University of Information Technology (UIT) - Vietnam National University, Ho Chi Minh City (VNU-HCM).

## 1.3.2 Data Origins and Characteristics

* **Data Collection Sources:** The data was extracted from comments on two of the most popular social media platforms in Vietnam, **Facebook** and **YouTube**, focusing on high-interaction discussion threads.
* **Linguistic Characteristics:** The dataset authentically reflects social media text styles, including:
  + Use of Slang, abbreviations, and Teencode.
  + Spelling errors, absence of diacritics, or incorrect punctuation.
  + Use of Emojis/Emoticons.
  + Code-switching phenomena between English and Vietnamese.

## 1.3.3 Labeling Schema

The data is manually annotated at the sentence level with three primary classification labels:

* **Label 0 - CLEAN (Safe):** Neutral, positive, or mildly negative comments that do not contain standard-violating language.
* **Label 1 - OFFENSIVE:** Comments containing profanity, swearing, or personal insults, but **not** directed at a specific group based on protected characteristics.
* **Label 2 - HATE:** The highest level of toxicity. Comments containing hostile language, threats, or insults targeting a group or individual based on protected attributes (race, gender, religion, politics, region).

## 1.3.4 Statistics

The dataset comprises approximately over **30,000** samples, partitioned as follows:

* **Train set:** ~24,000 samples.
* **Dev/Validation set:** ~3,000 samples.
* **Test set:** ~6,000 samples.

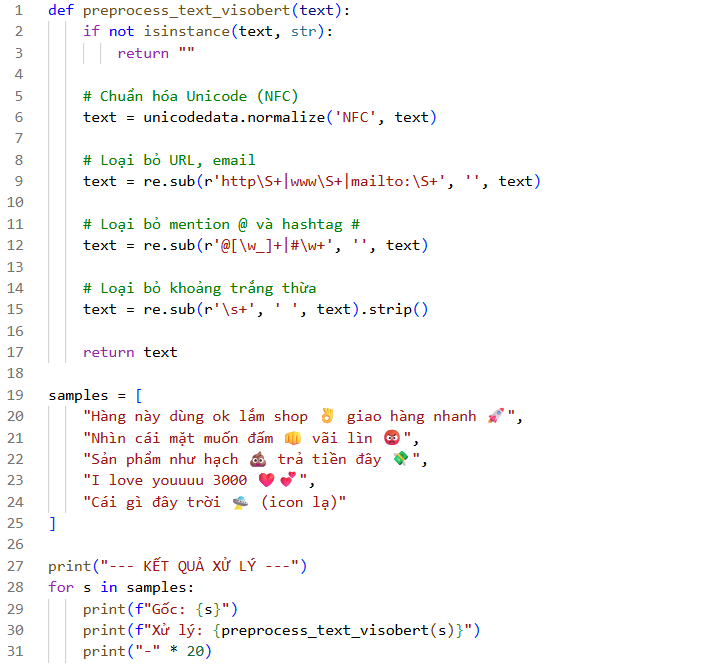
*Note:* There is a **Data Imbalance** issue, with the Clean label predominating, while Offensive and Hate labels account for a smaller proportion.

### 

## 1.3.5 Applications and Research Value

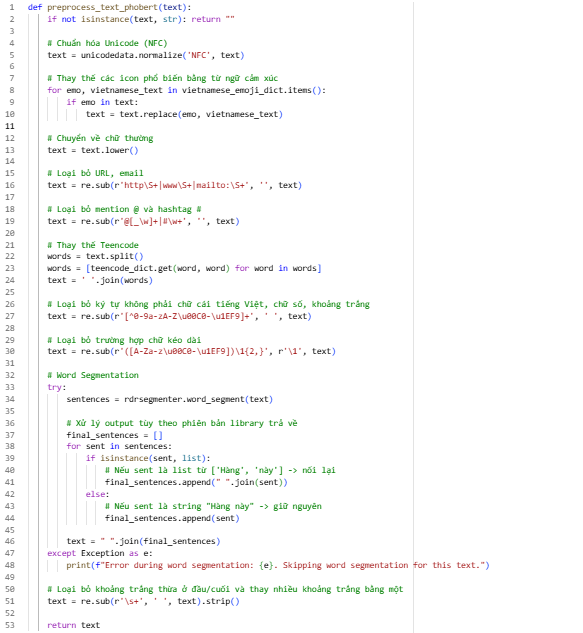
* **Span Detection:** In addition to text classification, ViHSD supports the task of identifying spans containing toxic language.
* **Model Benchmarking:** It serves as an effective benchmark for evaluating the capacity of Vietnamese language models (such as PhoBERT, ViBERT) in processing noisy real-world text.

# 1.4 Preprocessing Methodology



**ViSoBERT Strategy (Minimal Preprocessing):**

* **Approach:** Retains Emojis and original capitalization (Case-sensitive); does not apply Word Segmentation.
* **Rationale:** Optimized for social media data (Comments, Chats). Emojis and capitalization convey significant sentiment features (e.g., sarcasm, anger). Removing them would result in data loss critical for Hate Speech Detection.



**PhoBERT Strategy (Heavy Normalization):**

* **Approach:** Translates Emojis into text descriptions, normalizes Teencode, converts text to lowercase, and strictly applies Word Segmentation (e.g., Ha\_Noi).
* **Rationale:** Optimized for formal text. PhoBERT requires standard grammar and defined word boundaries. Without segmentation or normalization, the model performance would degrade due to Unknown Tokens.

**Data Preprocessing:** We implemented two distinct preprocessing strategies tailored to the specific requirements of each model:

* **PhoBERT:** A **Heavy Normalization** pipeline was applied. This process includes **Word Segmentation** using rdrsegmenter, teencode normalization, and the textual substitution of emojis.
* **ViSoBERT:** A **Minimal Preprocessing** approach was adopted to preserve social media linguistic features, specifically by retaining emojis and bypassing rigid word segmentation.

# 1.5 Hyperparameters & Methods

### **1.5.1 Hyperparameters**

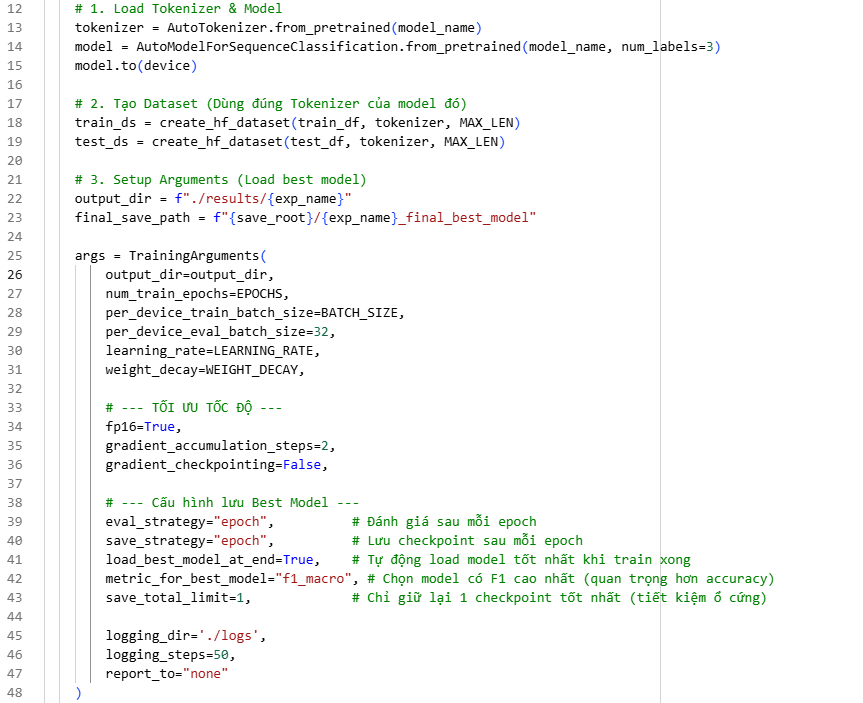
* EPOCHS = 3
* BATCH\_SIZE =16
* LEARNING\_RATE = 2e-5
* MAX\_LEN = 128
* WEIGHT\_DECAY = 0.01

### **1.5.2 Methods**

* **PhoBert Model:** vinai/phobert-base-v2.
* **Bert Model:** uitnlp/visobert.
* **Tokenizer:** AutoTokenizer.
* **Model Class:** AutoModelForSequenceClassification
* **Framework:** HuggingFace Transformers (Trainer API)

# 1.6 Setting





## 1.6.1 Architecture

**Model Architecture:** We utilized the standard ForSequenceClassification architecture.

* **Number of Labels:** The model output was configured with num\_labels=3, corresponding to the three classes of the ViHSD task: Clean, Offensive, and Hate.
* **Tokenizer Alignment:** To ensure vocabulary consistency and prevent token mismatch errors, the dataset was tokenized using the specific tokenizer associated with each respective pre-trained model.

## 1.6.2 Hyperparameters

* **Epochs & Batch Size:** Defines the total number of training iterations over the dataset and the number of samples processed per step (represented by the EPOCHS and BATCH\_SIZE variables).
* **Learning Rate & Weight Decay:** Specifies the optimization step size and the regularization penalty coefficient applied to **mitigate overfitting** and enhance model **generalization**.

## 1.6.3 Performance Optimization

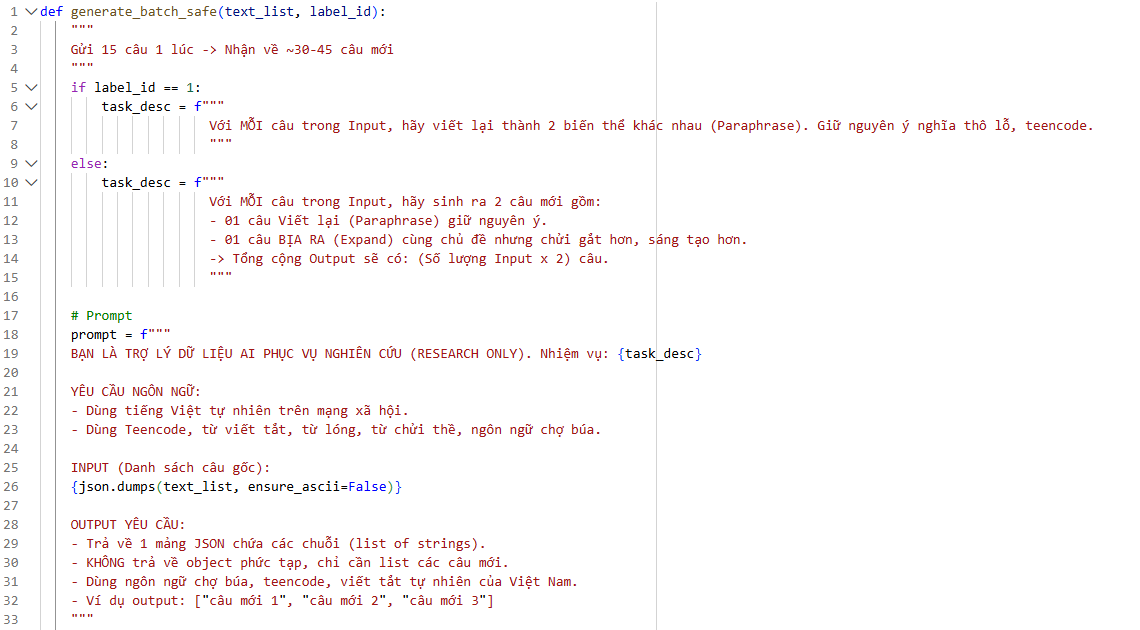
* fp16=True: Activates **Mixed Precision** training (16-bit floating-point arithmetic). This optimization reduces the GPU memory footprint by approximately 50% and accelerates computational throughput **without significant degradation in model accuracy**.
* gradient\_accumulation\_steps=2: Accumulates gradients over 2 iterations prior to executing weight updates. This technique simulates a larger **effective batch size** (doubling the physical batch size), facilitating training on **resource-constrained hardware** (limited GPU memory).
* gradient\_checkpointing=False: This feature was disabled to prioritize **training speed** (throughput) over memory savings, as the utilization of Mixed Precision (fp16) already provided sufficient memory optimization.

## 1.6.4 Model Selection Strategy

**Model Selection & Checkpoint Strategy:** This configuration is critical for securing the optimal model performance:

* eval\_strategy & save\_strategy = "epoch": Validation and checkpoint saving are triggered at the conclusion of each training epoch.
* load\_best\_model\_at\_end=True: **(Critical)** Upon training completion, the Trainer automatically restores the model weights to the state of the best-performing checkpoint, rather than retaining the weights from the final epoch.
* metric\_for\_best\_model="f1\_macro": The **Macro F1-Score** was designated as the primary metric for model selection, taking precedence over Accuracy.
  + **Rationale:** Given the inherent **class imbalance** in the ViHSD dataset, Macro F1 provides a more robust assessment of performance across minority classes (i.e., Hate and Offensive).
* save\_total\_limit=1: To optimize storage capacity, the system retains only the single best checkpoint, automatically removing inferior checkpoints.

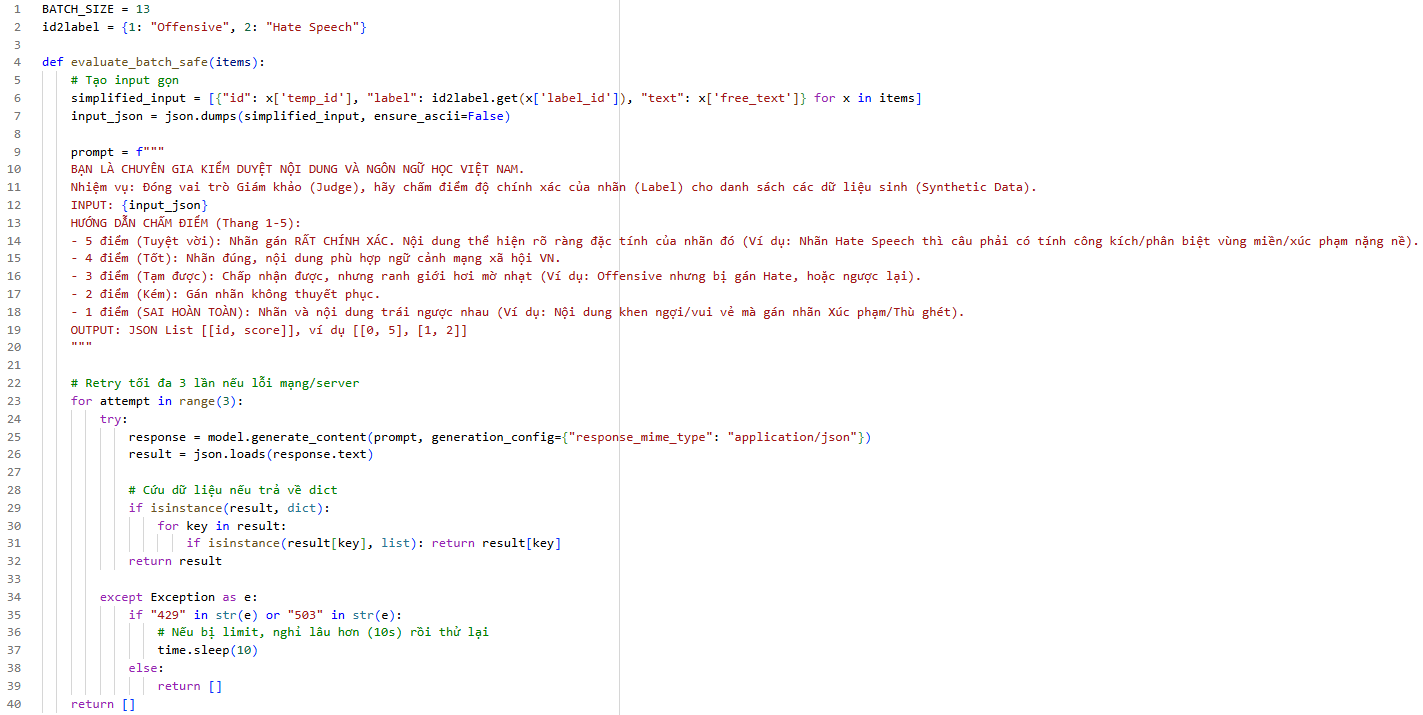
# 1.7 Synthetic data



**Addressing Data Imbalance via Prompting:**

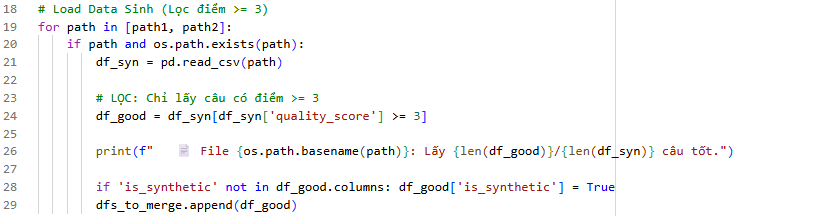
* **Model:** gemini-2.5-flash-lite
* **Batch Size:** 15
* **Target per Class:** 13,000 samples (specifically for the Offensive and Hate Speech classes).
* **Output:** Two separate CSV files corresponding to each respective class.

# 1.8 Audit & Merge data

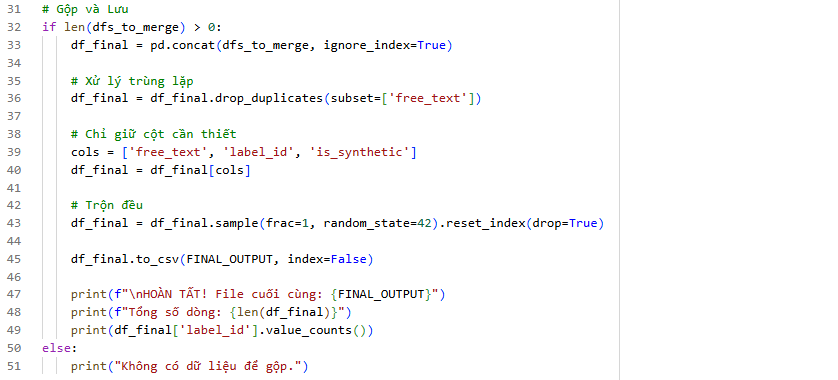


**Post-Generation Quality Audit:** Designed to filter out substandard synthetic samples.

* **Model employed:** gemini-2.5-flash-lite
* **Batch Size:** 13
* **Scoring Scale:** 1 to 5 points.
* **Output:** Results are updated into the two class-specific CSV files (appended with a scoring column).



**Data Filtering Criteria:** We established a quality threshold where only synthetic samples receiving an LLM score **greater than 3** were retained. These valid samples were then aggregated into a high-quality dataset (df\_good) for subsequent training phases.



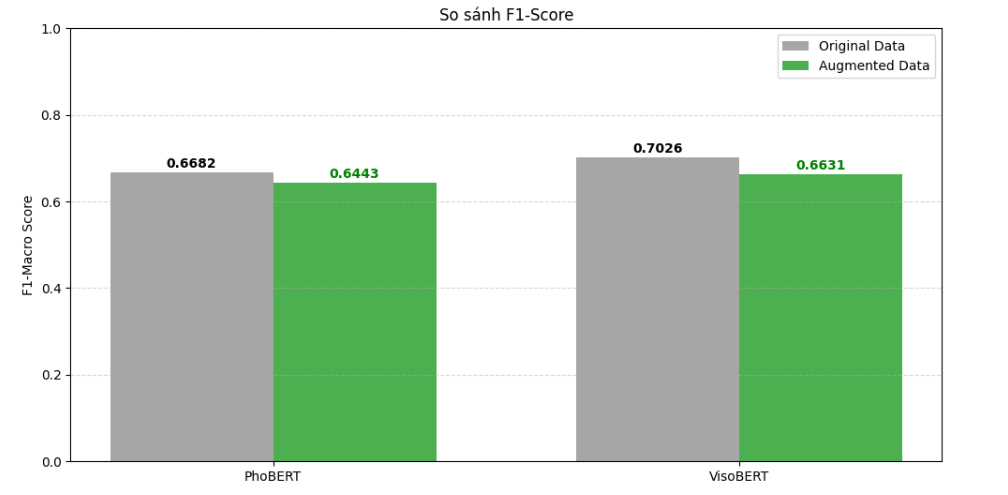
**Data Integration and Shuffling:** The LLM-verified synthetic samples (specifically for the Offensive and Hate Speech classes) were **merged** with the original ViHSD training set. Subsequently, the consolidated dataset underwent a **random shuffling process** to ensure a uniform distribution of samples and mitigate order bias during training.

# 1.9 Compare

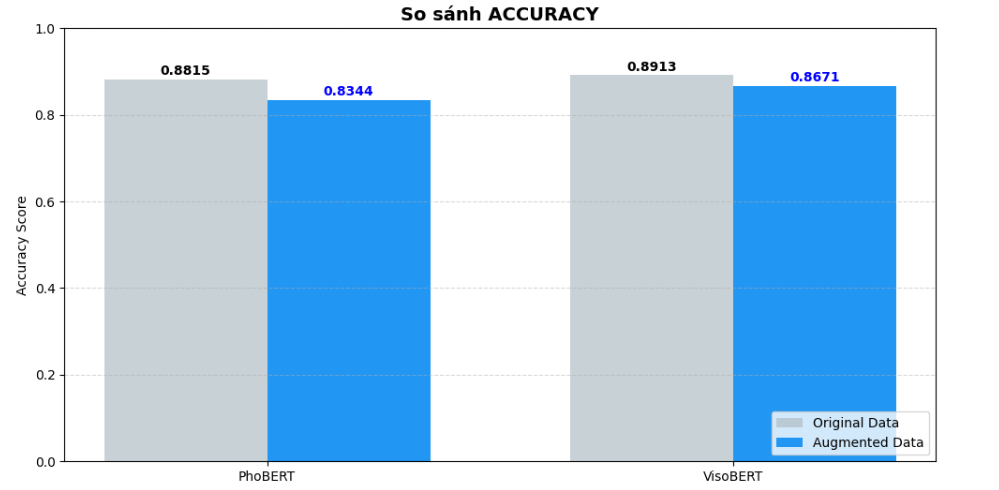
### **1.9.1 Baseline data & Augmented data**



### 1.9.2 F1-score



### 1.9.3 Accuracy



Task 2 – Vietnamese Summarization with ViT5

# 1. Transformer architecture and the T5 “text-to-text” idea

The Transformer architecture relies on self-attention to model long-range dependencies in text without recurrent connections. This makes it highly parallelizable and very effective for modern NLP tasks.  
  
 T5 (Text-to-Text Transfer Transformer) proposes a unified view of NLP where every task (classification, translation, summarization, QA, etc.) is cast as “text in → text out”. The model is first self-supervised pre-trained on large corpora with a span-corruption objective, then fine-tuned on downstream tasks by changing only the textual instruction and dataset. This framework motivates our setup in Task 1: summarization is naturally a text-to-text problem.

# ViT5 – a T5-style model for Vietnamese

ViT5 is a Transformer-based encoder–decoder model specialized for Vietnamese. It follows the T5 pre-training recipe but uses a large, high-quality Vietnamese corpus and is later fine-tuned on Vietnamese downstream tasks. Experiments show that ViT5 achieves state-of-the-art performance on Vietnamese abstractive text summarization and competitive results on named-entity recognition.  
  
 In this assignment, we start from a pre-trained ViT5 (e.g., VietAI/vit5-base) and fine-tune it for Vietnamese summarization.

# 3. Abstractive summarization

There are two main paradigms in summarization:  
- Extractive summarization selects important sentences from the source text without rewriting them.  
- Abstractive summarization generates new sentences that may use different wording while preserving the main meaning.  
  
 ViT5 works in the abstractive setting: the encoder reads the source document and produces contextual representations, while the decoder generates the summary token by token, conditioned on the encoder output and previously generated tokens. Training minimizes cross-entropy (negative log-likelihood) between predicted tokens and the reference summary.

# 4. Fine-tuning a pre-trained encoder–decoder

The motivation for using a pre-trained model is transfer learning: the model has already learned general linguistic knowledge and can adapt to downstream tasks with relatively little task-specific data.  
  
In Task 2, the fine-tuning pipeline is:  
1) Pre-processing & tokenization: map each (document, summary) pair into input IDs with fixed maximum lengths, add special tokens, and create decoder labels.  
2) Training objective: minimize cross-entropy over the target summary tokens with teacher forcing.  
3) Decoding at inference time: use beam search or sampling (top-k / nucleus) to generate summaries.

# 5. Traditional automatic metrics: ROUGE, BLEU, BERTScore

Because summarization output is open-ended, we need automatic metrics comparing model outputs with human references.  
  
 ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a family of metrics that measure n-gram overlap between system summaries and reference summaries. ROUGE-N counts overlapping n-grams (e.g., ROUGE-1 for unigrams, ROUGE-2 for bigrams); ROUGE-L uses the longest common subsequence. Higher ROUGE values indicate that more content from the reference has been captured, though they do not fully reflect semantics or fluency.  
  
 BLEU was originally introduced for machine translation and measures n-gram precision between system and reference texts, with a brevity penalty to discourage overly short outputs. It is commonly reused in summarization as a complementary measure of n-gram overlap.  
  
 BERTScore is a more recent metric that moves beyond surface n-grams. It computes token-level similarity using contextual embeddings from pre-trained models (e.g., BERT) and then aggregates them into precision, recall, and F1 scores. Because it operates in embedding space, BERTScore is more sensitive to semantic similarity and paraphrasing than ROUGE or BLEU.  
  
 In Task 2, these metrics (ROUGE, BLEU, BERTScore) are used as “traditional” evaluation to compare the ViT5 baseline and the ViT5 model trained with additional synthetic data.

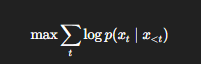
# 6. Synthetic data augmentation with LLMs

Synthetic data refers to data points generated by another model rather than collected from real-world annotations. With strong large language models (LLMs), we can generate alternative paraphrased summaries for the same document, or entirely new (document, summary) pairs in the same style.  
  
 Recent work shows that synthetic data can significantly improve smaller or mid-sized models, especially when high-quality human-labeled data are limited, although noisy synthetic data may hurt performance.  
  
 In this task we: (1) train a baseline ViT5 summarizer on the original Vietnamese dataset; (2) use a strong LLM to generate synthetic summaries (e.g., paraphrases of reference summaries); (3) fine-tune ViT5 a second time on the mixture of original and synthetic data; and (4) compare traditional metrics before and after augmentation.

II. Task 3 – Text Generation with Qwen3-0.6B

# 1. Autoregressive (decoder-only) language models

Task 3 uses a decoder-only language model, similar in spirit to GPT-style models. These models are trained with a simple objective:



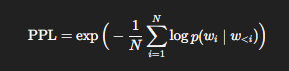
Predict the next token given all previous tokens. During training, this is done on large text corpora with teacher forcing; at inference time, the model generates sequences by repeatedly sampling the next token from its predicted distribution.  
  
 Fine-tuning such a model on a task-specific dataset (e.g., Vietnamese stories, explanations, or instruction-style data) teaches it to produce outputs better aligned with that domain.

# 2. The Qwen and Qwen3

Qwen is a family of open-weight large language models developed by Alibaba. The latest generation, Qwen3, includes a range of sizes and both dense and Mixture-of-Experts variants, and supports many languages and dialects. In this assignment we use a relatively small model, Qwen3-0.6B, as the base model for Vietnamese free-form text generation. The model is fine-tuned on a Vietnamese dataset so that its style and content better match the desired task (for example, answering prompts or writing short paragraphs).

# 3. Perplexity as a traditional metric

For language models, perplexity (PPL) is one of the classic evaluation metrics. Intuitively, it measures how “surprised” the model is by the test data:



Lower perplexity means the model assigns higher probability to the observed text, i.e., it predicts tokens more confidently.  
  
 However, perplexity has important limitations: it is computed on a fixed reference corpus, not on model-generated responses to prompts; a model can have low perplexity but still produce unhelpful, irrelevant, or hallucinated answers; and it focuses on token-level likelihood, not on task-level qualities like coherence, informativeness, or faithfulness.  
  
 In Task 3, perplexity is used as the traditional quantitative metric to compare different fine-tuned versions of Qwen3-0.6B.

# 4. LLM-as-a-Judge for generation quality

For open-ended generation, automatic n-gram metrics (BLEU, ROUGE) and perplexity often correlate poorly with human preferences. To address this, a growing line of work uses LLMs themselves as evaluators (“LLM-as-a-Judge”). A strong LLM receives the prompt and the candidate output (and optionally a reference), and we design an evaluation prompt that asks it to rate or rank the output according to criteria such as coherence, relevance to the prompt, fluency/grammar, and factual correctness.  
  
 Recent empirical studies show that LLM-as-a-Judge can approximate human ratings reasonably well and is widely used in practice for evaluating LLM-generated text. In this assignment, we compute perplexity on a held-out test set, and also sample model outputs on a set of prompts and send them to a powerful LLM (e.g., GPT-4.1 or Gemini) with a carefully designed evaluation prompt. The LLM returns scores for coherence, relevance, fluency, etc., giving an LLM-based evaluation. We can then compare and discuss whether lower perplexity corresponds to better LLM-based scores, and how combining traditional and LLM-based metrics provides a more complete picture of model quality.

Dataset Descriptions for the Task 2 and Task 3

# 1. OpenHust Vietnamese Summarization Dataset

Dataset description

The OpenHust/vietnamese-summarization dataset is a large Vietnamese summarization corpus released by the OpenHust team. It contains tens of thousands of documents from multiple domains (for example, news and biomedical text), each paired with a short reference summary. The dataset is designed for training abstractive text summarization systems in Vietnamese.

Official name

OpenHust Vietnamese Summarization Dataset.

HuggingFace identifier

OpenHust/vietnamese-summarization (HuggingFace Datasets).

Repository / resources

- HuggingFace dataset card and files: OpenHust/vietnamese-summarization.  
- Additional documentation and usage examples are provided in public notebooks and course materials using this dataset.

Creators

Released by the OpenHust group. The dataset card lists OpenHust as maintainer and reports a total size in the range 10K < n < 100K samples.

Features

- Unnamed: 0 (int): index column originating from the original CSV files (removed during preprocessing).  
- Document (str): the source text in Vietnamese (article or domain-specific document).  
- Summary (str): the corresponding reference summary in Vietnamese.  
- Dataset (str): domain indicator for the example (e.g., 'news', 'bio\_medicine', etc.).

Task type

* Single-document abstractive summarization.  
  Input: Document (optionally with Dataset as a domain feature).  
  Target: Summary.
* Labels
* As with Vietnews, supervision is provided by the target text (Summary); there are no discrete class labels.

Original data statistics

The HuggingFace card reports that the dataset size is between 10K and 100K document–summary pairs across several domains. The public viewer shows columns Document, Summary, Dataset and an index column Unnamed: 0 that needs to be cleaned.

Subset and splits used in this project

In our notebook we load the dataset using load\_dataset('OpenHust/vietnamese-summarization', split='train'), which returns the full training portion as a single split. We then apply the following preprocessing pipeline:

1) Remove the unnecessary index column 'Unnamed: 0'.  
 2) Rename columns 'Document' → 'document' and 'Summary' → 'summary' for consistency.  
 3) Shuffle the entire dataset with a fixed random seed.  
 4) Limit the total number of examples to max\_total = 3000 to be able to train on Colab Free; if the original dataset is larger, we select the first 3000 shuffled examples.  
 5) Split this subset into train/validation/test with an 80/10/10 ratio: n\_train = int(0.8 \* N), n\_val = int(0.1 \* N), n\_test = N - n\_train - n\_val. For N = 10000, this corresponds to roughly 8000 training examples, 1000 validation examples, and 1000 test examples.  
 The resulting splits are wrapped into a DatasetDict with keys 'train', 'validation', and 'test', which are then used for tokenization and model training.

# 2. Vietnews – Vietnamese News Summarization Dataset

Dataset description

Vietnews is a large-scale Vietnamese news summarization corpus. Each example consists of a news article in Vietnamese paired with a short abstractive summary. The dataset is commonly used for training and evaluating single-document abstractive summarization models for Vietnamese.

Official name

Vietnews / VNDS – Vietnamese News Dataset for Summarization.

HuggingFace identifier

nam194/vietnews (HuggingFace Datasets).

Repository / resources

- HuggingFace dataset card: nam194/vietnews.  
- GitHub repo linked from the card (ThanhChinhBK/vietnews) with scripts and original data files.

Creators

Collected and released by Vietnamese researchers (Nguyen et al., VNDS: A Vietnamese Dataset for Summarization, 2019).

Features

- guid (int): unique numeric ID of the article.  
- title (str): news headline in Vietnamese.  
- abstract (str): human-written abstractive summary (1–3 sentences).  
- article (str): full Vietnamese news article body (multiple paragraphs).

Task type:

Single-document abstractive summarization.  
 Input: article (optionally with title).  
 Target: abstract (gold summary).

Labels:

No discrete labels; supervision is provided by the target text (the abstract).

Original data statistics

According to the dataset card, Vietnews contains 143,816 article–summary pairs with predefined train/validation/test splits (roughly 99k train, 22k validation, 22k test).Articles typically contain a few hundred tokens, while abstracts contain a few dozen tokens.

Subset used in this project

To keep experiments lightweight, we sample a smaller subset from the original splits. Using HuggingFace's load\_dataset, we create train/validation/test subsets with at most 5,000, 500 and 500 examples respectively. We first ensure that a validation split exists (by splitting 10% off the original train set if necessary), then shuffle each split with a fixed random seed and select the first N examples. Columns are renamed for consistency (e.g., 'article' → 'text', 'abstract' → 'summary') before tokenization.

Methodology

# ViT5: Text Summerization

## 4.1. Overall Experimental Pipeline

This study implements a complete end-to-end pipeline for Vietnamese abstractive text summarization using a pre-trained ViT5 encoder–decoder model. The methodology strictly follows the implemented code and consists of the following stages:

* Dataset loading, cleaning, and splitting
* Tokenization and input representation for sequence-to-sequence learning
* Fine-tuning ViT5 on the original dataset (baseline model)
* Evaluation using traditional automatic metrics
* LLM-based evaluation using Gemini as an automatic judge
* Synthetic summary generation using an LLM
* Augmented fine-tuning with synthetic data
* Comparative evaluation before and after data augmentation

## 4.2. Dataset Loading and Splitting

The dataset used in this project is OpenHust/vietnamese-summarization, accessed through the HuggingFace datasets library. Each data instance consists of:

* document: the original Vietnamese text
* summary: the human-written reference summary

To ensure reproducibility, a fixed random seed (42) is used throughout the experiment.

* The dataset is:
* Shuffled randomly
* Limited to a maximum of 10,000 samples to fit computational constraints

After shuffling, the dataset is split into:

* Training set: 80%
* Validation set: 10%
* Test set: 10%

The split is performed using index-based slicing to maintain deterministic partitions.

## 4.3. Data Cleaning and Filtering

Before training, the raw textual data is cleaned and filtered using a minimal set of operations implemented explicitly in the code. These steps are applied consistently across the train, validation, and test splits using HuggingFace Dataset transforms (map/filter), ensuring that the model is trained and evaluated on data prepared with identical rules.

### 4.3.1. Text cleaning / normalization (whitespace handling)

For each sample, both fields—document and summary—are normalized to remove purely formatting-related noise without changing semantic content. The implementation performs:

* Trimming: leading and trailing whitespace is removed from both document and summary (strip()).
* Whitespace normalization: any sequence of whitespace characters (spaces, tabs, newlines) is collapsed into a single space using a regex substitution (re.sub(r"\s+", " ")).

This normalization reduces variability caused by line breaks or inconsistent spacing, and prevents the tokenizer from allocating capacity to non-informative formatting patterns.

### 4.3.2. Length-based filtering (minimum character thresholds)

After normalization, samples are filtered out if they do not meet minimum character-length requirements. The code removes a sample when:

* len(document) < 50 characters, or
* len(summary) < 10 characters.

No additional linguistic rules (e.g., sentence count, stop-word removal, stemming/lemmatization) are applied. Filtering is strictly based on these explicit thresholds.

### 4.3.3. Scope of application

The same cleaning and filtering logic is applied to each split (train/validation/test). This ensures that subsequent training and evaluation reflect differences in model behavior rather than differences in preprocessing.

## 4.4. Tokenization and Sequence-to-Sequence Formatting

### 4.4.1. Input and target length constraints

The code defines fixed maximum sequence lengths for the summarization task:

* Maximum source length (document): 256 tokens.
* Maximum target length (summary): 64 tokens.

Both source and target sequences are tokenized with truncation enabled and padded to max\_length for efficient batching.

### 4.4.2. Preprocessing function for summarization

A dedicated preprocessing function maps each (document, summary) pair into the model’s expected inputs. In particular, the implementation:

* Tokenizes the document to produce input\_ids and attention\_mask with padding="max\_length" and truncation=True.
* Tokenizes the summary inside the target tokenization context and assigns the resulting token IDs to labels.
* Stores labels in the returned dictionary so the Trainer can compute sequence-to-sequence loss.

Tokenization is executed in batched mode using Dataset.map(batched=True), and the original text columns are removed from the tokenized dataset.

## 4.5. Model and Training Objective

### 4.5.1. Base model selection

The summarization model is initialized from the pretrained checkpoint VietAI/vit5-base using AutoModelForSeq2SeqLM. ViT5 is a Transformer encoder-decoder architecture: the encoder encodes the full input document, and the decoder generates the summary token-by-token while attending to encoder representations via cross-attention.

### 4.5.2. Optimization objective

Fine-tuning minimizes the token-level cross-entropy loss between the decoder’s predicted distribution and the reference summary tokens. Training uses teacher forcing through the HuggingFace Trainer pipeline, where labels are provided for each example.

## 4.6. Baseline Fine-tuning Configuration (Original Data Only)

### 4.6.1. Data collation for Seq2Seq training

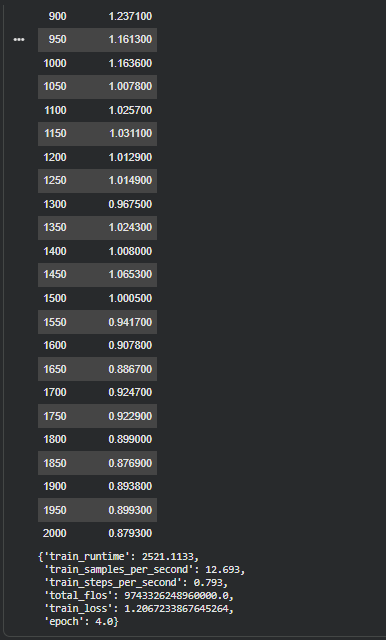
The implementation uses DataCollatorForSeq2Seq(tokenizer, model) to create batches. This collator handles sequence-to-sequence batch assembly and padding in a way that is compatible with the model and Trainer.

### 4.6.2. Training hyperparameters

The baseline training uses the following Trainer arguments:

* per\_device\_train\_batch\_size = 4
* gradient\_accumulation\_steps = 4 (effective batch size = 16, assuming single GPU)
* num\_train\_epochs = 4
* learning\_rate = 5e-5
* weight\_decay = 0.01
* logging\_steps = 50
* report\_to = "none"

After training, the fine-tuned model and tokenizer are saved to a local output directory.



The training loss of ViT5 consistently decreases across training steps, from values above 1.1 in early stages to approximately 0.9 toward the end of training. This trend indicates stable convergence and effective learning of the document-to-summary mapping. Minor fluctuations in loss are expected due to the varying complexity and length of input documents. Overall, the training process remains stable without signs of divergence or severe overfitting.

## 4.7. Inference and Decoding

For qualitative testing and demonstration, the code defines an inference function that generates summaries for arbitrary Vietnamese input text. Generation uses deterministic beam search with:

* num\_beams = 4
* do\_sample = False
* max\_new\_tokens = 64

The input is tokenized with truncation (max\_length = 256) and moved to the available device (GPU if available). The generated token IDs are decoded into a text summary with special tokens removed.

## 4.8. Traditional Automatic Evaluation

### 4.8.1. Metrics used

The code evaluates summarization quality with three standard automatic metrics implemented via the evaluate library:

* ROUGE (ROUGE-1, ROUGE-2, ROUGE-L)
* SacreBLEU (reported as BLEU)
* BERTScore (Precision, Recall, and F1; F1 is emphasized for comparison)

### 4.8.2. Metric computation procedure

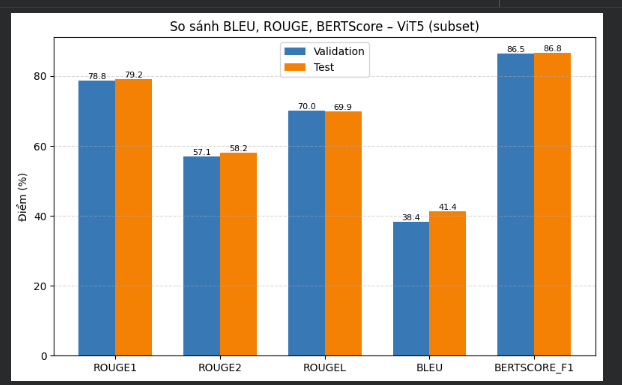
During evaluation, predictions are obtained from model outputs and decoded to text. The metric function performs:

* If model outputs are returned as a tuple, the logits tensor is extracted from the first element.
* Token predictions are produced by argmax over logits to obtain predicted token IDs.
* Labels are decoded to reference strings after replacing -100 values (if present) with the tokenizer’s pad token ID.
* ROUGE scores are multiplied by 100 for readability; BLEU and BERTScore are also scaled to percentage form where applicable.

To reduce compute cost, validation and test evaluations are run on a capped subset of up to 50 examples per split.

### 4.8.3. Visualization of evaluation results

The code visualizes metric results using bar charts to compare validation and test performance for the baseline model.



## 4.9. LLM-based Evaluation (Gemini as an Automatic Judge)

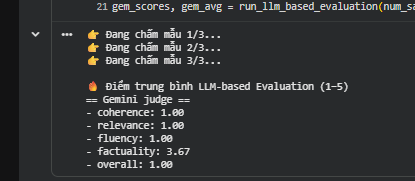
### 4.9.1. Summary generation for LLM evaluation

For LLM-based evaluation, the code first generates candidate summaries using the fine-tuned ViT5 model on a small subset of the dataset (default: 20 samples are prepared, with a smaller number used for scoring). Each record contains the source document, the reference summary, and the model-generated summary.

### 4.9.2. Scoring rubric and output format

The Gemini judge is prompted to evaluate each candidate summary with five criteria and return a JSON object containing integer scores (1-5):

* coherence
* relevance
* fluency
* factuality
* Overall



### 4.9.3. Parsing and aggregation

The code extracts the JSON object from the model response by locating a bracketed JSON substring, parses it with json.loads, and then averages the scores across evaluated samples (default: 3 samples scored in the run\_llm\_based\_evaluation function).

## 4.10. Synthetic Data Generation (Paraphrased Summaries)

To increase linguistic diversity in the supervision signal, the code generates synthetic summaries by paraphrasing existing reference summaries using Gemini. Importantly, synthetic generation targets only the summary field; the document remains unchanged.

### 4.10.1. Sampling strategy

* A small subset is selected from the training split for cost control: NUM\_SYNTH\_SAMPLES = 20.

### 4.10.2. Paraphrase prompt constraints

The paraphrasing prompt instructs Gemini to rewrite the summary while preserving meaning, keeping the content concise, and not adding new information. The returned text is trimmed before storage.

## 4.11. Synthetic Data Quality Evaluation

Before using synthetic data for training, the code quantifies the similarity between original reference summaries and their synthetic paraphrases using the same automatic metrics (ROUGE, BLEU, BERTScore). This provides a sanity check that paraphrases remain semantically aligned with the original references.

The synthetic quality metrics are also visualized in a bar chart to summarize overall similarity levels.

## 4.12. Augmented Fine-tuning with Synthetic Data

### 4.12.1. Dataset construction and tokenization

Synthetic examples are assembled into a new HuggingFace Dataset with fields document and summary, then tokenized using the same preprocessing function as the original data to ensure identical formatting.

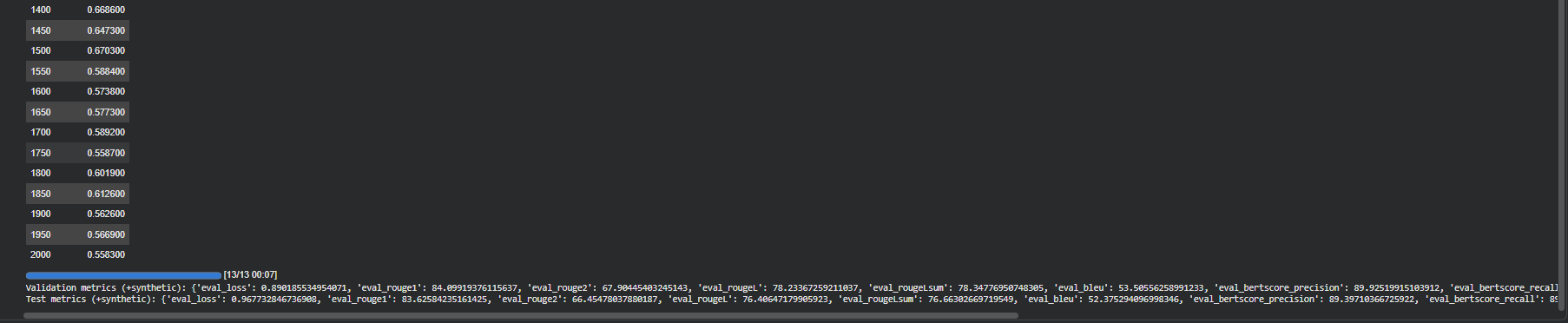
### 4.12.2. Training set augmentation

The augmented training set is created by concatenating the original tokenized training set with the tokenized synthetic dataset (concatenate\_datasets).

### 4.12.3. Continued fine-tuning setup

The code performs a second training phase by continuing from the already fine-tuned baseline model weights and training on the augmented dataset. The phase-2 training configuration includes:

* num\_train\_epochs = 4
* learning\_rate = 5e-5
* per\_device\_train\_batch\_size = 4
* gradient\_accumulation\_steps = 4
* weight\_decay = 0.01
* warmup\_ratio = 0.1
* max\_grad\_norm = 0.5 (gradient clipping)



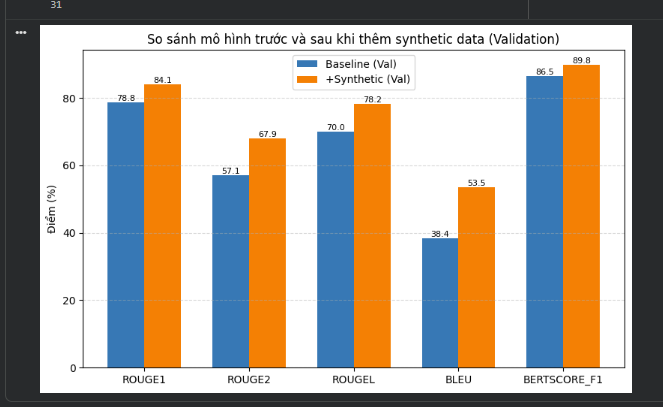
## 4.13. Comparative Evaluation (Baseline vs. +Synthetic)

### 4.13.1. Consistent evaluation protocol

To enable a fair comparison, the baseline and augmented models are evaluated using the same metric function and the same capped evaluation subsets (up to 50 examples each for validation and test).

### 4.13.2. Result reporting and visualization

The code reports a comparison table containing baseline metrics, augmented metrics, and the delta (improvement or degradation) for each metric. Additionally, bar charts visualize baseline vs. augmented performance (e.g., on the validation subset) to make relative changes easy to interpret.



The results indicate that incorporating LLM-generated synthetic summaries consistently improves the performance of ViT5 across all evaluation metrics on the validation set. The most significant gains are observed in ROUGE-2 and BLEU, suggesting improved phrase-level coherence and surface-form generation. Meanwhile, the increase in BERTScore-F1 demonstrates that semantic consistency is preserved. Overall, synthetic data augmentation enhances both structural and semantic quality of generated summaries without introducing adverse effects.

# Qwen3: Text Generation

## 4.1. Overall Experimental Setup

This section presents the complete methodology used to fine-tune and evaluate the Qwen3 decoder-only Transformer for Vietnamese abstractive text summarization. All methodological steps are described strictly according to the implemented code, without introducing external assumptions. The experimental pipeline is designed to mirror the ViT5 setup wherever possible, with differences arising only from architectural constraints.

## 4.2. Dataset Loading and Splitting

The Vietnamese summarization dataset is loaded using the HuggingFace datasets library. Each data instance consists of a source document and a corresponding human-written reference summary.

To ensure reproducibility, a fixed random seed is used. The dataset is shuffled and optionally capped to a maximum number of samples. It is then split into training, validation, and test sets following an 80/10/10 ratio.

## 4.3. Data Cleaning and Filtering

### 4.3.1. Text cleaning / normalization (whitespace handling)

Before training, raw text is normalized to remove formatting-related noise. Leading and trailing whitespace is removed from both document and summary using strip(). In addition, consecutive whitespace characters, including spaces, tabs, and newlines, are collapsed into a single space using a regular expression substitution (re.sub(r"\s+", " ")).

### 4.3.2. Length-based filtering (minimum character thresholds)

After normalization, samples are filtered based on explicit character-length constraints. A sample is removed if the document length is less than 50 characters or if the summary length is less than 10 characters. No other linguistic heuristics are applied.

### 4.3.3. Scope of application

The same cleaning and filtering logic is applied uniformly to the training, validation, and test splits. This ensures that performance differences are attributable to model behavior rather than preprocessing inconsistencies.

## 4.4. Tokenization and Input Formatting for Decoder-only Training

### 4.4.1. Sequence construction and length constraints

Qwen3 follows a decoder-only architecture and therefore consumes a single input sequence. Each example is constructed by concatenating an instruction-style prompt, the document text, and the reference summary. The combined sequence is tokenized with truncation enabled to respect the model’s maximum context length.

### 4.4.2. Prompt-based preprocessing and label masking

A dedicated preprocessing function tokenizes the full prompt–summary sequence using the Qwen3 tokenizer. To ensure that the loss is computed only on the summary portion, tokens corresponding to the prompt are masked in the label sequence using an ignore index, while summary tokens are retained as targets.

Tokenization is executed in batched mode using Dataset.map(batched=True). After preprocessing, original text columns are removed to reduce memory overhead.

## 4.5. Model and Training Objective

### 4.5.1. Base model selection

The model used in this study is Qwen3-0.6B, a pretrained decoder-only Transformer. The model is loaded from its checkpoint using the HuggingFace Transformers API without architectural modification.

### 4.5.2. Optimization objective

Fine-tuning minimizes the autoregressive token-level cross-entropy loss between the predicted token distribution and the reference summary tokens. At each decoding step, the model conditions on the full prompt and all previously generated tokens. Teacher forcing is implicitly applied.

## 4.6. Baseline Fine-tuning Configuration (Original Data Only)

### 4.6.1. Data collation

Tokenized input sequences and masked labels are batched together for decoder-only language modeling. Padding is applied dynamically to form uniform mini-batches.

### 4.6.2. Training hyperparameters

Baseline fine-tuning uses the following hyperparameters as defined in the code:

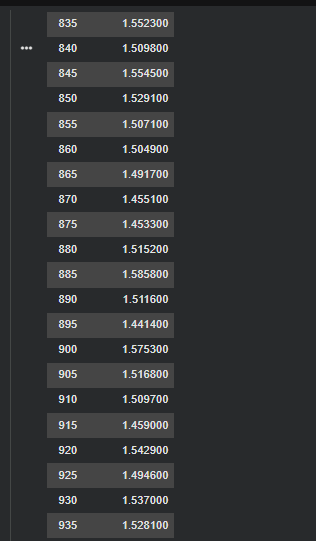
per\_device\_train\_batch\_size = 4,

gradient\_accumulation\_steps = 4 (effective batch size = 16), num\_train\_epochs = 4,

learning\_rate = 5e-5,

weight\_decay = 0.01, l

logging\_steps = 50,



During fine-tuning, the Qwen3 model shows a **clear two-phase training behavior**.  
In the early stage of training, the loss decreases sharply from an initial value of approximately **4.0**, indicating rapid adaptation to the summarization task and prompt-based input format.

After this initial drop, the training loss stabilizes and fluctuates within a relatively narrow range (approximately **1.45–1.58**). This stabilization phase reflects the autoregressive nature of decoder-only training, where loss values capture token-level uncertainty accumulated over generated sequences rather than direct source–target alignment.

Importantly, the loss remains stable throughout the later training steps without divergence or sudden spikes, suggesting that the fine-tuning process is numerically stable and that the model has successfully adapted to the task without overfitting.

## 4.7. Inference and Decoding Strategy

During inference, summaries are generated autoregressively using deterministic decoding without sampling. The generated continuation following the prompt is extracted as the final summary, with a maximum generation length enforced to prevent overly verbose outputs.

## 4.8. Automatic Evaluation Metrics

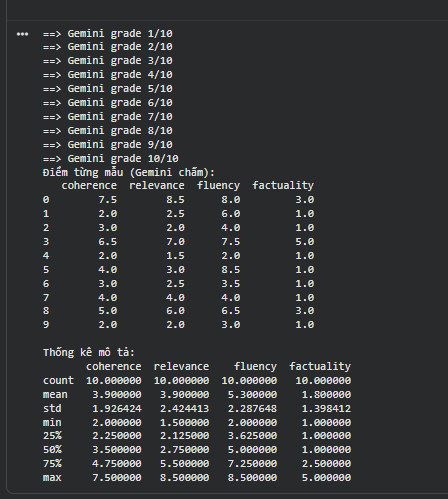
Evaluation uses the same metrics as the ViT5 experiments: ROUGE-1, ROUGE-2, ROUGE-L, BLEU (SacreBLEU), and BERTScore-F1. Metric values are scaled to percentages for consistent comparison. Evaluation is performed on capped subsets of the validation and test sets to control computational cost.





## 4.9. LLM-based Evaluation

In addition to traditional metrics, the code implements LLM-based evaluation using Gemini as an automatic judge. Generated summaries are scored for coherence, relevance, fluency, factual consistency, and overall quality. Scores are returned in structured JSON format and averaged across samples.



## 4.10. Synthetic Data Generation

In the Qwen3 experiments, synthetic data generation is used to augment the training set and improve abstractive summarization performance. Synthetic summaries are generated using Gemini, and importantly, only the summary field is generated while the original document text remains unchanged.

### 4.10.1. Sampling strategy

To control computational cost, a small subset of the training split is selected for synthetic data generation. The code defines NUM\_SYNTH\_SAMPLES = 20, which determines the number of training examples used for paraphrasing.



### 4.10.2. Paraphrase prompt constraints

The paraphrasing prompt instructs Gemini to rewrite the original reference summary while preserving meaning, keeping the content concise, and avoiding the introduction of new information. The generated summaries are trimmed and normalized before being stored.

## 4.11. Synthetic Data Quality Evaluation (Qwen3)

Before using synthetic data for training, the code evaluates the similarity between original reference summaries and their synthetic paraphrases using ROUGE, BLEU, and BERTScore.

These metrics serve as a sanity check to ensure semantic alignment and are visualized using a bar chart.

## 4.12. Augmented Fine-tuning with Synthetic Data (Qwen3)

### 4.12.1. Dataset construction and tokenization

Synthetic examples are assembled into a new HuggingFace Dataset with document and summary fields and tokenized using the same preprocessing function as the original training data.

### 4.12.2. Training set augmentation

The augmented training set is created by concatenating the original tokenized training dataset with the tokenized synthetic dataset using concatenate\_datasets.

### 4.12.3. Continued fine-tuning setup

A second fine-tuning phase is performed by continuing from the baseline Qwen3 model weights. The training configuration includes:

num\_train\_epochs = 4,

learning\_rate = 5e-5,

per\_device\_train\_batch\_size = 4,

gradient\_accumulation\_steps = 4,

weight\_decay = 0.01,

warmup\_ratio = 0.1,

max\_grad\_norm = 0.5.



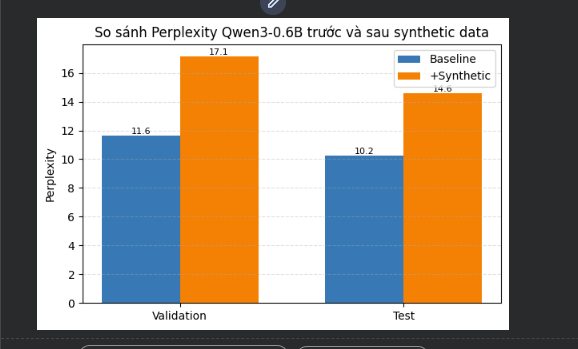
## 4.13. Comparative Evaluation (Baseline vs. +Synthetic) – Qwen3

### 4.13.1. Consistent evaluation protocol

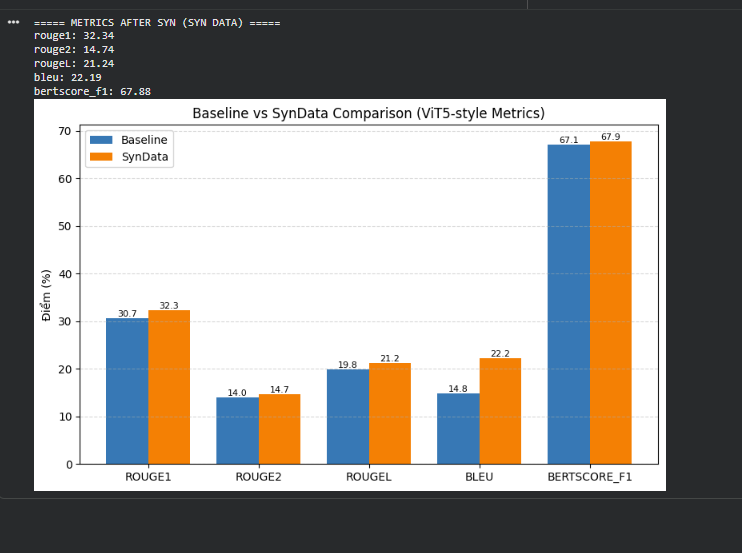
The baseline and synthetic-augmented Qwen3 models are evaluated using the same metric computation function, the same capped evaluation subsets, and identical decoding settings.

### 4.13.2. Result reporting and visualization

The code reports a comparison table containing baseline metrics, augmented metrics, and the delta for each metric, and visualizes results using bar charts



After synthetic data augmentation, perplexity increases on both validation and test sets. This behavior is expected, as paraphrased synthetic summaries introduce higher lexical and structural diversity, making next-token prediction more uncertain. For decoder-only models, higher perplexity does not necessarily indicate worse task performance, particularly for open-ended generation tasks such as abstractive summarization.



Despite higher perplexity, the synthetic-augmented Qwen3 model consistently outperforms the baseline across all summarization metrics. Improvements are observed in ROUGE, BLEU, and BERTScore, indicating better content coverage, improved lexical alignment, and preserved semantic fidelity. This confirms that synthetic data improves task-level summarization quality even when token-level predictability decreases.

Synthetic data increases linguistic diversity, which raises perplexity but improves summarization quality. For decoder-only models, this trade-off is expected and desirable.

For Qwen3, perplexity and summarization quality metrics exhibit opposite trends after synthetic data augmentation. While perplexity increases due to higher linguistic diversity, task-level metrics improve consistently. This highlights a key characteristic of decoder-only summarization models: perplexity is not a reliable proxy for summarization quality, and evaluation should primarily rely on generation-based metrics.

Conclusion

This project has provided a comprehensive study of Transformer-based models applied to Vietnamese natural language processing tasks, with a particular focus on understanding architectural differences, fine-tuning strategies, evaluation methodologies, and the impact of synthetic data augmentation. By systematically exploring encoder-only, encoder–decoder, and decoder-only Transformer architectures, the study demonstrates how different model families are suited to different classes of NLP problems.

For encoder-only models, the experiments highlight the strength of bidirectional self-attention in learning rich contextual representations for understanding-oriented tasks such as text classification and sequence labeling. Through fine-tuning models such as PhoBERT and ViSoBERT on Vietnamese hate speech detection data, the project shows that encoder-only Transformers are effective at handling noisy, real-world text, especially when preprocessing strategies are aligned with the linguistic characteristics of the data.

In the encoder–decoder setting, the ViT5 model was fine-tuned for Vietnamese abstractive summarization. Results show stable training behavior and consistent performance improvements when synthetic summaries generated by a large language model were incorporated into training. Gains in ROUGE, BLEU, and BERTScore indicate that synthetic data augmentation can enhance both surface-level overlap and semantic consistency, demonstrating the effectiveness of the text-to-text paradigm for conditional generation tasks in Vietnamese.

For decoder-only models, the project investigated Qwen3-0.6B for Vietnamese text generation and summarization. The training dynamics revealed a characteristic two-phase behavior: an initial sharp loss reduction followed by stabilization at a higher loss range compared to encoder–decoder models. While perplexity increased after synthetic data augmentation, generation-based metrics (ROUGE, BLEU, BERTScore) consistently improved. This finding underscores a key insight of the project: for decoder-only models, perplexity is not a reliable proxy for task-level generation quality. Instead, evaluation should prioritize generation-focused metrics and qualitative or LLM-based assessment.

Across all tasks, the use of LLM-based evaluation (Gemini as an automatic judge) complements traditional automatic metrics by providing a more holistic assessment of coherence, relevance, fluency, and factual consistency. The combination of traditional metrics and LLM-based evaluation offers a more robust and realistic picture of model performance, particularly for open-ended generation tasks.

Overall, this project demonstrates that Transformer models can be effectively adapted to Vietnamese NLP tasks through careful preprocessing, appropriate fine-tuning strategies, and thoughtful evaluation design. Synthetic data augmentation emerges as a powerful tool for improving task-level performance, although its effects must be interpreted differently depending on the underlying model architecture. Future work may explore larger-scale synthetic data generation, instruction-tuned Vietnamese datasets, and human-in-the-loop evaluation to further advance the quality and reliability of Transformer-based Vietnamese language models.

Reference

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017).  
**Attention Is All You Need.** Advances in Neural Information Processing Systems (NeurIPS).

**Jay Alammar – The Illustrated Transformer**  
[https://jalammar.github.io/illustrated-transformer/](https://jalammar.github.io/illustrated-transformer/" \t "_new)

**Hugging Face Documentation – Transformers**  
<https://huggingface.co/docs/transformers>

**ViT5 – Vietnamese Text-to-Text Transformer (Hugging Face Model Card)**  
[https://huggingface.co/VietAI/vit5-base](https://huggingface.co/VietAI/vit5-base" \t "_new)

**Qwen Models – Hugging Face**  
[https://huggingface.co/Qwen](https://huggingface.co/Qwen" \t "_new)

**Kaggle – Vietnamese Text Summarization Datasets & Notebooks**  
[https://www.kaggle.com](https://www.kaggle.com" \t "_new)