VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY



NLP FINAL PROJECT

Transformer Based Encoder – Decoder And GPT Model For Medical Q&A

*Lecturer*: Assoc. Prof. Dr. LE ANH CUONG

*Student*: NGUYEN KHAC HUY – 521H0502

GIANG HOANG DAT – 521H0498

TRAN DINH QUANG VINH – 521H0333

*Class* : 21H50301

Group : 9

*Academic year* : 2024-2025

HO CHI MINH CITY, 2025

VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY



NLP FINAL PROJECT

Transformer Based Encoder – Decoder And GPT Model For Medical Q&A

*Lecturer*: Assoc. Prof. Dr. LE ANH CUONG

*Students*: NGUYEN KHAC HUY – 521H0502

GIANG HOANG DAT – 521H0498

TRAN DINH QUANG VINH – 521H0333

*Class* : 21H50201

Group : 9

*Course* : 2024 - 2025

HO CHI MINH CITY, 2025

ACKNOWLEDGEMENT

To complete this final project, we would like to express our gratitude to Ton Duc Thang University for providing the necessary facilities and to the teachers who have supported us throughout our studies at the TDT University. Moreover, we love to give thanks to our parents who gave us many chances to approach knowledge at TDTU and motivated, supported us a lot.

Especially, we would like to thank Assoc. Prof. Dr. Le Anh Cuong for teaching us with great dedication and detail, so that we have enough knowledge that we used for this essay. Due to our limited experience and acknowledgement, we make sure that there are some mistakes in our work. We sincerely hope to receive feedback and constructive criticism from the teacher who instructed us. So that we can complete this essay more effectively.

We would like to express our heartfelt thanks and wishes the teacher’s good health.

# THE PROJECT IS COMPLETED AT TON DUC THANG UNIVERSITY

I hereby declare that this is the product of our project and is guided by Pham Van Huy. The research contents and results in this topic are honest and have not been published in any forms before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data from other authors and other organizations, all with citations and source notes.

If any fraud is discovered, I will take full responsibility for the content of my project. Ton Duc Thang University is not involved in copyright violations caused by me during the implementation process (if any).

*Ho Chi Minh City, 8th May 2025*

*Authors*

*(Signatures and full names)*

*Trần Đình Quang Vinh*

*Giang Hoằng Đạt*

*Nguyễn Khắc Huy*

SUMMARY

This report delves into the integration of Reinforcement Learning (RL) in Natural Language Processing (NLP), particularly in enhancing the capabilities of Large Language Models (LLMs). It begins by introducing the foundational concepts of RL, explaining how decision-making frameworks can be applied to language model optimization. Key algorithms such as Proximal Policy Optimization (PPO) and Direct Preference Optimization (DPO) are explored in detail. Their mechanisms, advantages, and limitations are compared to highlight their suitability for different NLP tasks, especially those requiring alignment with human preferences.

The second chapter focuses on the dataset used in this study and its preparation using Byte Pair Encoding (BPE), a widely used tokenization method in NLP. It outlines the criteria for dataset selection and presents the preprocessing steps necessary for effective subword segmentation. The chapter also provides a theoretical overview of BPE and details its practical implementation, including training the BPE model to convert raw text into a format suitable for training language models.

In the third chapter, the report turns to Transformer-based architectures, beginning with an explanation of the encoder-decoder structure that underpins many modern NLP systems. The process of building a Transformer model from scratch is described, along with the fine-tuning of the T5 model for specific applications. Attention then shifts to GPT models, where both the architecture of GPT and the methods for fine-tuning GPT-2 are discussed in the context of medical question answering. This chapter emphasizes the adaptability of pre-trained models for domain-specific tasks.

The final chapter is dedicated to evaluating and comparing the performance of the models using ROUGE and BLEU scores. These metrics assess the quality of generated text by comparing it with reference outputs, measuring aspects such as precision, recall, and fluency. The chapter presents the evaluation results, allowing for a comparative analysis of different model architectures and training strategies.

TABLE OF CONTENTS

[ACKNOWLEDGEMENT 1](#_Toc197799919)

[THE PROJECT IS COMPLETED AT TON DUC THANG UNIVERSITY 2](#_Toc197799920)

[SUMMARY 3](#_Toc197799921)

[TABLE OF CONTENTS 5](#_Toc197799922)

[CHAPTER I – ReInforment Learning In NLP with LLMs 7](#_Toc197799923)

[1.1. Introduction of RL 7](#_Toc197799924)

[1.2. Theoretical of RL 9](#_Toc197799925)

[1.2.1. RL Algorithms 9](#_Toc197799926)

[1.2.2. Proximal Policy Optimization Algorithm(PPO) 14](#_Toc197799927)

[1.2.3. Direct Preference Optimization (DPO) 17](#_Toc197799928)

[1.2.4. Compare algorithms 19](#_Toc197799929)

[CHAPTER II – Exploring Dataset And Byte Pair Encoding (BPE) 22](#_Toc197799930)

[2.1 Dataset for requirement 22](#_Toc197799931)

[2.1.1. Information of dataset 22](#_Toc197799932)

[2.1.2. Preprocessing dataset for BPE 24](#_Toc197799933)

[2.2. Theory of Byte Pair Encoding (BPE) 26](#_Toc197799934)

[2.3. Setting up and Training BPE 28](#_Toc197799935)

[CHAPTER III – Transformer And GPT Model For Medical Q&A 34](#_Toc197799936)

[3.1 Transformer based Encoder – Decoder architecture 34](#_Toc197799937)

[3.2 Building Transformer model from scratch and Fine – tuning T5 model 35](#_Toc197799938)

[3.3. GPT model architecture 35](#_Toc197799939)

[3.4 Creating GPT model from scratch and Fine – tuning GPT2 35](#_Toc197799940)

[CHAPTER IV – Evaluation And Compare Models With Rouge And BLEU Score. 36](#_Toc197799941)

[REFERENCES 37](#_Toc197799942)

CHAPTER I – ReInforment Learning In NLP with LLMs

Introduction to Reinforcement Learning algorithms used in LLMs

Reinforcement Learning (RL) is a field within machine learning that examines how agents can learn to make decisions through trial and error in order to maximize cumulative rewards. Through interaction with an environment and feedback based on their actions, machines can learn via RL. This feedback manifests as either rewards or penalties.

Numerous algorithms have been explored and utilized in the process of refining large language models (LLMs) through Reinforcement Learning. The shared aim is to train a text-generating model (like GPT) in such a way that its outputs align with user expectations or receive high praise from a reward model. REINFORCE, Proximal Policy Optimization (PPO), and Direct Preference Optimization (DPO) are the three most popular and influential algorithms in RL for LLMs.

1.1. Introduction of RL

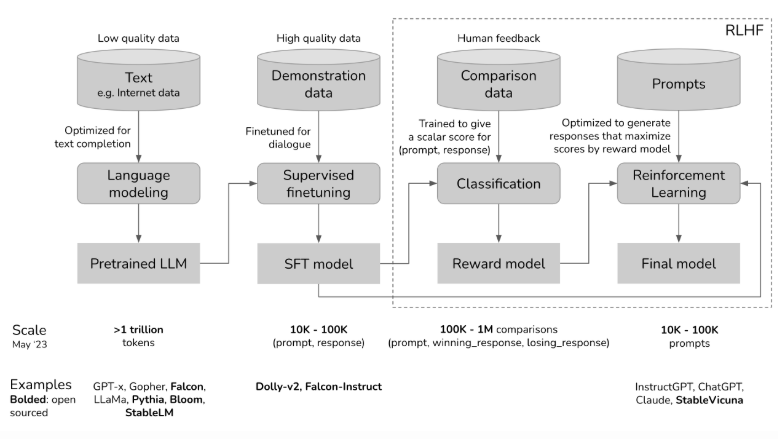
Recently, large language models (LLMs) like GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers), and T5 (Text-To-Text Transfer Transformer) have advanced significantly in natural language processing tasks, including text generation, question answering, summarization, machine translation, and others. These models utilize the Transformer architecture to efficiently learn semantic relationships between words in sentences, having been trained on billions of words from large and diverse data sets. Specifically, GPT serves as a robust autoregressive text generation model; BERT aims to comprehend context via word masking; and T5 recasts all NLP tasks as text-to-text generation.

Still, the predominant methods for training LLMs are supervised learning and unsupervised pretraining. Although these methods are effective, they come with numerous limitations: the model may produce incorrect answers, lack ethical appropriateness, or diverge from user intent. To this end, Reinforcement Learning (RL) has been proposed as a supplementary approach to fine-tune the text generation model based on user or expert feedback.

Reinforcement Learning from Human Feedback (RLHF) is a common application of RL in LLM. The RLHF process involves three stages: (1) Pretraining the language model on extensive text datasets, (2) Fine-tuning the reward model with labeled comparisons of responses, and (3) Fine-tuning the text generator model (like GPT) through a reinforcement learning algorithm, typically REINFORCE or Proximal Policy Optimization (PPO). A representative case is ChatGPT from OpenAI, which is refined through RLHF to enhance the safety, politeness, and utility of its responses.

Utilizing RL in LLM provides multiple benefits compared to relying solely on supervised learning. To begin with, RL enables models to learn from intricate and qualitative goals like “answer well,” “be contextually appropriate,” or “follow user instructions.” These objectives are challenging to encapsulate through supervised learning. Secondly, models can use RL to adjust continuously based on real-world feedback and enhance user experience. Furthermore, RL aids in preventing models from being influenced by training data that is frequently biased, inaccurate, or does not truly represent user intent.

The integration of LLM and RL, particularly via RLHF, is enhancing the capability, dependability, and amicability of intelligent dialogue systems. Research focused on incorporating RL into the training of language models will remain significant for the future of language AI.

*Figure 1.1. RLHF sample workflow.*

1.2. Theoretical of RL

1.2.1. RL Algorithms

REINFORCE is a policy gradient algorithm that is both simple and straightforward. It refreshes the policy by maximizing the anticipated reward. Within the framework of LLM, the policy πθ indicates the likelihood of the model producing a text string y in response to a prompt x.

Parameter update formula θ:

Where:

: Gradient of the objective function by parameter of policy . The goal is to maximize , usually the expected reward.

: Expected value on trajectory = ( , , , , ,…, , , ,

: A complete trajectory of states (s), actions (a), and rewards (r) from the initial time to the end (or a certain time threshold T).

: Probability of trajectory τ when obeying policy

: Sum over all time steps t in a trajectory.

: Gradient of the log probability of performing the action

in state by parameter

: Total rewards received throughout trajectory τ, usually calculated as

This formula indicates that the gradient of the objective function can be approximated by averaging the product of the log probability gradient of each action taken in a trajectory and the total reward received in that trajectory. We can enhance the policy to achieve greater rewards by calculating this gradient and adjusting the parameter θ in the direction of the increasing gradient.

Meaning: The model will increase the likelihood of producing sequences that yield high rewards, while reducing the likelihood of those that yield low rewards.

**Gradient of the target**

Meaning: This is the main formula of the REINFORCE algorithm.

It allows to calculate the gradient to update the model parameters in the direction of increasing the probability of generating high-reward responses.

If the response 𝑦 has a reward > 0, then the gradient increases ; if 𝑅 < 0 the gradient decreases it.

**Calculate token gradient (autoregressive)**

Meaning: Since the language model generates each word/token sequentially, the total gradient can be decomposed into the sum of the gradients at each generation step.

This makes training more efficient in frameworks like PyTorch or TensorFlow.

**Add baseline (reduce variance)**

Meaning: When we subtract a mean value b(x), the gradient is still correct but has lower variance → the model is easier to converge.

b(x): the baseline can be the mean reward, or the output of the value function model.

**Loss function**

Meaning: This is the actual loss function used for training.

We minimize this loss with optimizer → which is equivalent to maximizing the reward.

Each generated token will be penalized or rewarded depending on the total feedback reward.

**Reward from reward model**

Meaning: Rewards are not always available from users, so reward models are often used to evaluate the quality of feedback.

Reward models are trained from preference data or rankings provided by reviewers.

**Sampling sequence**

(y|x)

Meaning: The model must sample the actual output from its distribution to calculate the reward.

Don't use greedy decoding, but rather use sampling, to ensure the gradient follows the probability distribution.

**How Reinforcement Learning Works?**

In the process of RL, an agent takes actions within an environment and receives rewards or penalties depending on those actions. The agent then modifies its behavior in response. This loop aids the agent in enhancing its decision-making over time with the aim of maximizing the cumulative reward.

Here’s a breakdown of RL components:

Policy: A strategy that the agent uses to determine the next action based on the current state.

Reward Function: A function that provides feedback on the actions taken, guiding the agent towards its goal.

Value Function: Estimates the future cumulative rewards the agent will receive from a given state.

Model of the Environment: A representation of the environment that predicts future states and rewards, aiding in planning.

**Types of Reinforcements in RL**

1. Positive Reinforcement

Positive Reinforcement is characterized by an event that happens as a result of specific behavior and serves to enhance the strength and frequency of that behavior. Put differently, it influences behavior positively.

Advantages: Performance is maximized, and change can be maintained over time.

Disadvantages: Excessive use can result in states of excess that might diminish effectiveness.

2. Negative Reinforcement

Negative Reinforcement refers to the process of strengthening behavior by stopping or avoiding a negative condition.

Advantages: Raises the frequency of behavior, ensures a baseline level of performance.

Disadvantages: It could motivate only the minimum amount of action necessary to evade punishment.

**Application of Reinforcement Learning**

Robotics: RL finds application in automating tasks within structured settings like manufacturing, enabling robots to learn movement optimization and efficiency enhancement.

Game Playing: Strategies for intricate games such as chess, Go, and video games have been crafted using advanced RL algorithms, which have surpassed human players on numerous occasions.

Industrial Control: RL assists in the real-time fine-tuning and optimization of industrial activities, including processes in the oil and gas sector.

Customized Training Systems: RL allows for the tailoring of instructional materials to fit a person's unique learning behaviors, which enhances both engagement and efficacy.

**Advantages of Reinforcement Learning**

Solving Complex Problems: RL is capable of solving highly complex problems that cannot be addressed by conventional techniques.

Error Correction: The model continuously learns from its environment and can correct errors that occur during the training process.

Direct Interaction with the Environment: RL agents learn from real-time interactions with their environment, allowing adaptive learning.

Handling Non-Deterministic Environments: RL is effective in environments where outcomes are uncertain or change over time, making it highly useful for real-world applications.

**Disadvantages of Reinforcement Learning**

Not Suitable for Simple Problems: RL is often an overkill for straightforward tasks where simpler algorithms would be more efficient.

High Computational Requirements: Training RL models requires a significant amount of data and computational power, making it resource-intensive.

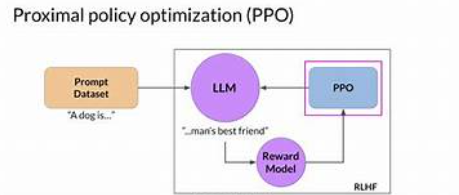
Dependency on Reward Function: The effectiveness of RL depends heavily on the design of the reward function. Poorly designed rewards can lead to suboptimal or undesired behaviors.

Difficulty in Debugging and Interpretation: Understanding why an RL agent makes certain decisions can be challenging, making debugging and troubleshooting complex

Reinforcement Learning is a robust method for making decisions and optimizing in dynamic environments. The intricacy of RL, however, requires that reward functions be designed with care and that considerable computational resources be allocated. RL can be utilized to address complex real-world challenges and promote progress in multiple sectors by understanding its foundational concepts and uses.

1.2.2. Proximal Policy Optimization Algorithm(PPO)

PPO is a contemporary and effective RL algorithm, commonly utilized in RLHF (e.g., ChatGPT). PPO makes the model more stable during training by limiting the rate of policy change through a penalty function (clipping), rather than updating the policy too aggressively like REINFORCE.

*Figure 1.2. PPO*

PPO loss function formula:

Where:

is the ratio of the probability of acting under the new policy to the old policy.

is the benefit value (advantage) at the time t

is a hyperparameter to control the range of the ratio

This means that, by steering clear of major updates at every step, PPO enhances the stability and efficiency of the learning model compared to REINFORCE.

PPO has demonstrated significant effectiveness across a variety of applications, including AI model training, gaming, and robot control in intricate settings. Below are several remarkable benefits of PPO:

High effectiveness in acquiring optimal policies.

Easier to deploy and more stable than other algorithms like DDPG or TRPO.

Exhibits effective generalization across various environments.

PPO is extensively utilized in numerous studies and practical applications, playing a major role in the advancement of reinforcement learning.

**How PPO Works**

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm designed to improve the stability and efficiency of policy optimization methods. Here are the detailed steps of how PPO works:

1. Initialisierung der Start-Policy: Zunächst legen wir eine Policy zufällig oder auf der Grundlage früherer Erfahrungen fest.

2. Execute the policy to gather data: The existing policy facilitates interaction with the environment and data collection regarding actions and rewards. This dataset includes state-action pairs along with their respective rewards.

3. Calculate the advantage value for each action. This value assesses the disparity between the actual reward and the predicted reward.

4. Revise the policy: Employ PPO’s specialized loss function for policy revision. This loss function aims to prevent excessive policy changes, thus aiding stability maintenance. The formula for the loss function is:

Where:

is the ratio of the probability of acting under the new policy to the old policy, calculated by the formula:

Where:

is the value of the benefit at the time

is a hyperparameter to control the range of the ratio

This loss function uses clipping to restrict the value of , thereby preventing excessively large changes in the policy and maintaining stability throughout learning.

This process is reiterated multiple times until either the optimal policy is identified or a specified performance threshold is achieved.

==> PPO has shown to be efficient and stable across a variety of applications, such as gaming, robot control, and other intricate optimization challenges.

Advantages and disadvantages of PPO

Proximal Policy Optimization (PPO) is a widely used reinforcement learning algorithm due to its efficiency and stability. Here are the main advantages and disadvantages of PPO:

- Advantages of PPO

+ High efficiency: PPO can learn optimal policies effectively, aiding in the maximization of rewards obtained from the environment.

+ Stability: The PPO employs algorithm a unique loss function that includes clipping to limit the size of policy updates, thereby ensuring a stable learning process.

+ Easy to implement: Due to its simplicity compared to other reinforcement learning algorithms like TRPO or DDPG, PPO is easier to implement and modify.

+ Strong generalization capability: PPO can be utilized and adapted across a variety of contexts and situations, including gaming, robot control, and process optimization.

+ Batch optimization: With its capability for batch optimization, PPO enhances computational efficiency and maximizes the use of hardware resources.

- Disadvantages of PPO

+ Hyperparameter dependence: The effectiveness of PPO depends heavily on the choice of hyperparameters such as learning rate and clip parameters. Tuning these hyperparameters can require a lot of experimentation and tuning.

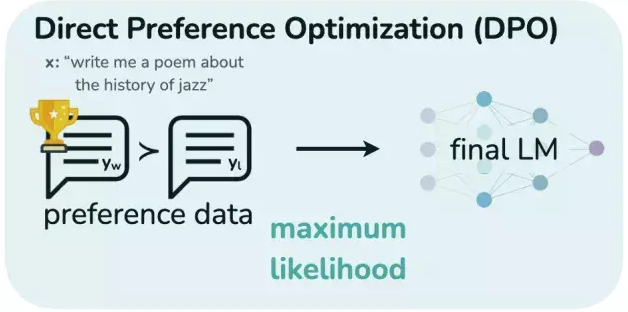
+ Computational resource requirements: PPO, like many other reinforcement learning algorithms, requires a large amount of computational resources for processing and training, especially when applied in complex environments.

+ Not optimal in all cases: Although PPO works well in many situations, it is not always the best choice. In some specific cases, other algorithms may yield better results.

1.2.3. Direct Preference Optimization (DPO)

DPO, which was developed in 2023, is an algorithm that aims to train language models based on pairwise comparisons of answers instead of using absolute scores. DPO directly enhances the likelihood that the model produces the superior response in the pair (y\_winner, y\_loser), eliminating the necessity for a reward model to be used as an intermediary.

DPO (Direct Preference Optimization) is a method for training language models that learns directly from human preferences data, bypassing the need for a reward model or reinforcement learning algorithms such as PPO.

*Figure 1.3. DPO.*

**Optimal formula:**

Where:

: is the language model being trained with parameter , which outputs the probability of response when given prompt .

: is a reference model (usually a pre-trained model or a model that has undergone initial supervised fine-tuning). Using a reference model helps ensure that the model does not stray too far from its original capabilities.

(z) = : is sigmoid function

: is a hyperparameter that controls how important it is to optimize according to preferences versus staying close to the reference model. Higher values of will make the model follow the preference data more strongly, while lower values will keep the model closer to ​

**DPO Algorithm:**

Data Preparation: Collect a dataset of preference comparison pairs, where each sample consists of a prompt, a preferred response, and a less preferred response for that prompt.

Model Initialization: Start with a pre-trained or supervised fine-tuning language model (). This model will be fine-tuned using the DPO algorithm to produce the model

Loss Calculation: For each pair () in the dataset, calculate the DPO loss value based on the above formula. This requires calculating the log probability of both the preferred response and the less preferred response under both the current model and the reference model

Parameter Update: Use gradient descent (or another optimization algorithm) to update the model parameters to minimize the loss value. The gradient of the loss function can be calculated based on the derivative of the log sigmoid function.

Iteration: Repeat steps 3 and 4 over the entire dataset (or batches) until the model converges or the desired performance is achieved.

**What DPO Means:**

Direct Preference-Based Optimization: : DPO adjusts the language model directly to raise the likelihood of responses favored by humans and lower the likelihood of those less favored, as opposed to optimizing indirectly via a distinct reward model such as RLHF.

Process Simplification: DPO removes the complicated training phase for reward models and reinforcement learning algorithms, which leads to a fine-tuning process that is simpler and more stable.

Computational Efficiency: DPO typically requires less computational resources and shorter training times than RLHF.

Competitive or Better Performance: Studies have shown that DPO can achieve comparable or even better performance than RLHF in tuning LLMs to human preferences on a variety of tasks such as dialogue generation, text summarization, and sentiment control.

Easy to deploy: With simple training objectives based on binary classification, DPO easily integrates into existing training workflows.

1.2.4. Compare algorithms

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **REINFORCE** | **Direct Preference Optimization (DPO)** | **Proximal Policy Optimization (PPO)** |
| **Algorithm Type** | Policy Gradient (Monte Carlo) | Preference-based Optimization | Policy Gradient (Actor-Critic) |
| **Objective** | Directly optimize the policy to achieve the highest reward | Directly optimize the policy based on human preferences | Optimize the policy while ensuring the stability of the training process |
| **Input Data** | Complete episodes (state, action, reward) | Preference comparison pairs (prompt, preferred response, less preferred response) | Trajectories (state, action, reward) and value function estimates |
| **Policy Update** | Update policy after the end of an episode (Monte Carlo) | Update policy based on each preference comparison pair | Update policy in small batches (mini-batches) with a trust region mechanism (through clipping) |
| **Complexity** | Relatively simple to understand and implement | Simpler than RLHF (no need to train a reward model) | More complex than REINFORCE but generally more stable |
| **Stability** | Can have high variance due to updates based on the entire episode | Generally more stable than RLHF due to direct optimization based on preference | High stability due to the clipping mechanism that limits policy changes |
| **Main Applications** | Simple tasks, easy to simulate with clear episode endings | Fine-tuning large language models (LLMs) to align with human preferences | Continuous control tasks, games with complex action spaces |

*Table 1. Comparison of RL algorithms.*

CHAPTER II – Exploring Dataset And Byte Pair Encoding (BPE)

In this chapter, we will briefly present the dataset, which is also the place to present details about word tokenization algorithms such as byte pair encoding. In more detail, in this chapter, we will present the theory of the BPE algorithm and apply it to the dataset just presented.

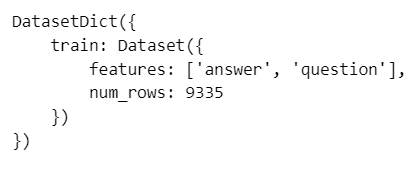
2.1 Dataset for requirement

2.1.1. Information of dataset

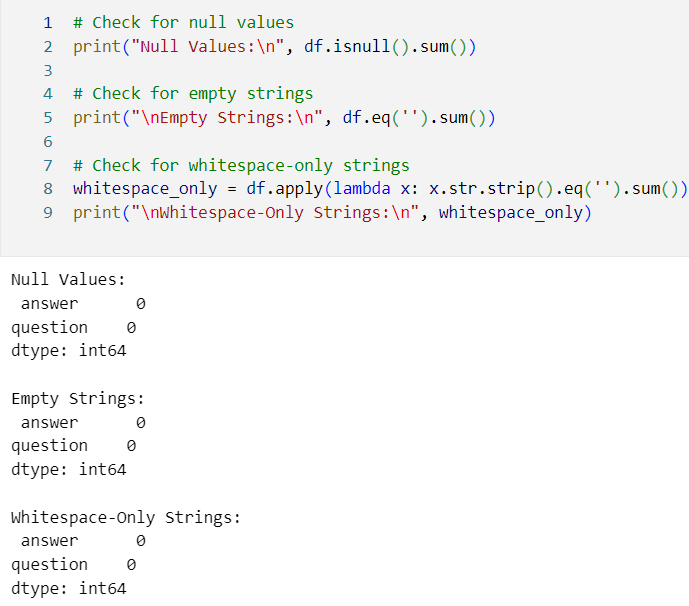
We will introduce the dataset used in this report. The dataset is taken from Huggingface, a place that specializes in providing data, algorithms, and models for natural language processing.

The dataset used is named "vietnamese-medical-qa" provided by the author "hungmn" on 16/02/2024. According to the author's description, this dataset is compiled from edoctor and vinmec.

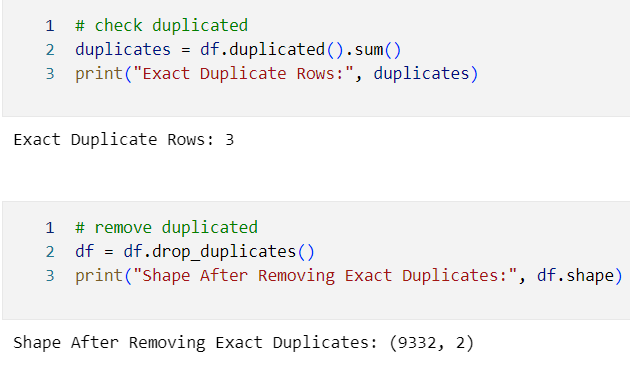
The link of the dataset will be annotated in the code section of the report. The following will be a description of the data and the process we used to clean and preprocess it.

*Figure 2.1. Construction of dataset.*

As we have presented, this dataset has two columns, **answer** and **question**. With 9335 pairs of q&a, this could be a dataset that is neither too large nor too small to build a medical support agent in our report.

*Figure 2.2. Checking null values of dataset.*

We will check if the data contains null values ​​or not, maybe if the data contains only a few values ​​then it will not affect too much but if it is the opposite then it will affect the results of the model later. Luckily, our data choose not to have null values.

*Figure 2.3. Checking duplicated in dataset.*

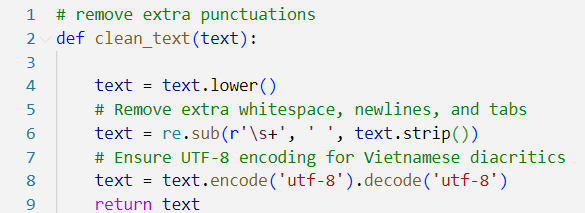
Check if the data is duplicated or not, if there is a duplicate we will delete it because it can cause interference because those words appear more in the vocabulary.

2.1.2. Preprocessing dataset for BPE

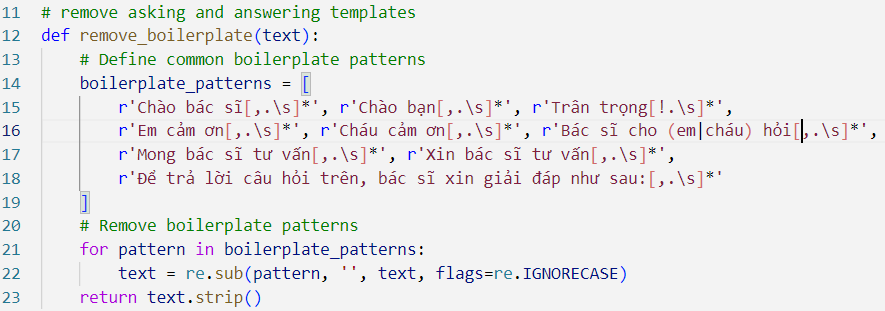
In the data preprocessing step, we will use the underthesea library as shown in the image below because we work with Vietnamese data. Because Vietnamese will have some semantics and contexts that are different from languages ​​like English, and this library supports very well in separating words in sentences so that they are reasonable in terms of word meaning.



*Figure 2.4. Packages for data preprocessing.*

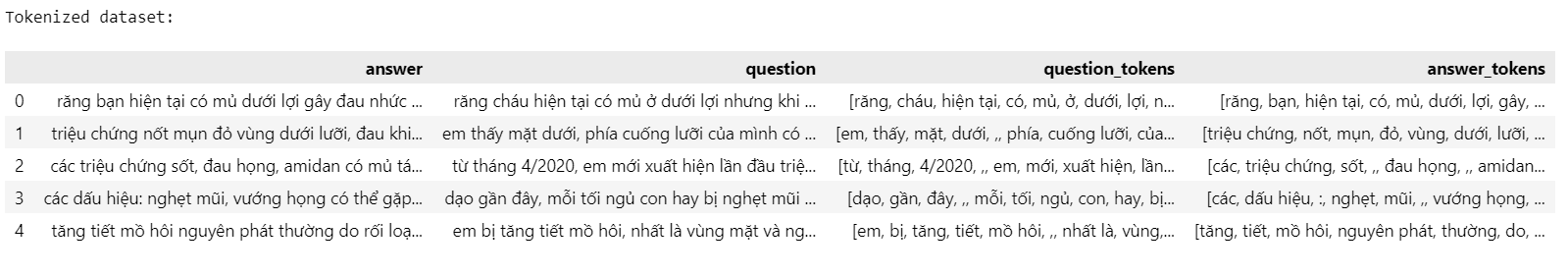
We have two important functions in this section, clean\_text and remove\_boilerplate. These functions have the following functions respectively, the clean\_text function has the function of cleaning, lowercase and removing unnecessary spaces and newlines, the next function has the function of removing the templated response prompts as shown in the code, removing the templates in the questions and answers of the data to make the model focus more on the medical information, which is what we are aiming for.

*Figure 2.5. Cleaning function for dataset.*

**

*Figure 2.6. Function to remove boilerplate in questions and answers.*

Then we will apply these functions to the dataset, below is the result. For easier processing, this dataset will be saved to drive via google colab.

*Figure 2.7. Dataset after applying these functions and word\_torkenize.*

2.2. Theory of Byte Pair Encoding (BPE)

Byte – Pair Encoding (BPE) is a text tokenization technique in Natural Language Processing. It breaks down words into smaller, meaningful pieces called subwords. It works by repeatedly finding the most common pairs of characters in the text and combining them into a new subword until the vocabulary reaches a desired size. This technique helps in handling rare or unknown words by breaking them into smaller parts that the model has already learned during training.

Key elements to understanding BPE:

* **Vocabulary**: In BPE vocabulary refers to the set of subword units (tokens) used to represent all the words in the corpus. After applying BPE, vocabulary consists of all the subwords that can be used to represent a word in the dataset.
* **Byte**: It is a basic unit of digital information that consists of 8 bits. It is used to represent characters that are encoded as bytes that will be merged.
* **Character**: It is a single letter, number or punctuation mark in text. We start by treating each word as a sequence of characters and merging process creates subword units based on character pairs.
* **Frequency**: It refers to how many times a particular byte or character appears in the corpus. BPE is driven by the frequency of character pairs meaning that the most frequent pair of characters will be merged first.
* **Merge**: It is the process of combining two consecutive characters or subword units into a new unit. Each merge reduces the total number of tokens in the corpus and leads to more abstract representations of words.

Now lets break down step by step to clearify how BPE works.

1. *Initialize the Vocabulary.*

* Start with the input text corpus.
* Treat each character in the text as an initial token.
* Add a special end-of-word token (in this report we use </w>) to mark word boundaries if working with subword units for NLP.
* Create a frequency table of all tokens in the corpus.

**Example:** Input text: "low low lower lowest".

* Split into words and add *</w>*:[*"l o w </w>", "l o w </w>", "l o w e r </w>", "l o w e s t </w>"*]*.*
* Initial vocabulary: { *"l", "o", "w", "e", "r", "s", "t", "</w>"* }.
* Token frequency: Count how often each character appears in the sequence.

1. *Count Pair Frequencies*

* Identify all adjacent pairs of tokens in the tokenized corpus.
* Calculate the frequency of each pair (how often each pair appears consecutively).

**Example:** Tokenized corpus: [*"l", "o", "w", "</w>", "l", "o", "w", "</w>", "l", "o", "w", "e", "r", "</w>", "l", "o", "w", "e", "s", "t", "</w>"*].

* Adjacent pairs:
  + "l o": appears 4 times (once in each word).
  + "o w": appears 4 times.
  + "w </w>": appears 2 times (in "low" twice).
  + "w e": appears 2 times (in "lower" and "lowest").
  + "e r": appears 1 time (in "lower").
  + "e s": appears 1 time (in "lowest").
  + "s t": appears 1 time (in "lowest").
  + "r </w>": appears 1 time (in "lower").
  + "t </w>": appears 1 time (in "lowest").

1. *Merge the Most Frequent Pair.*

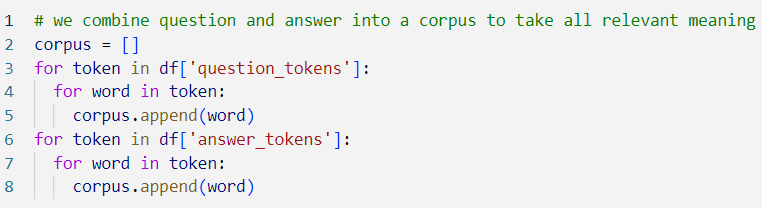
* Identify the most frequent pair of tokens in the corpus.
* Merge this pair into a single new token (e.g., combine "l" and "o" into "lo").
* Update the vocabulary by adding the new token.
* Replace all occurrences of the pair in the corpus with the new token.

**Example:**

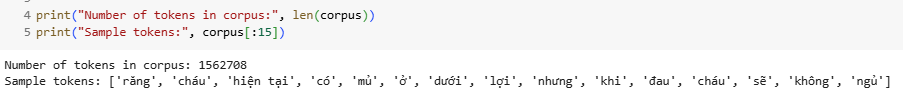
* Most frequent pairs: "*l o*" and "*o w*" (both appear 4 times). Let’s choose "*l o*" arbitrarily.
* Merge "*l o*" into "*lo*".
* Update vocabulary: { *"l", "o", "w", "e", "r", "s", "t", "</w>", "lo"* }.
* Update corpus: Replace "l o" with "lo" in all words:
  + "*l o w </w>*" → "*lo w </w>*" (for both "low" instances).
  + "*l o w e r </w>*" → "*lo w e r </w>*".
  + "*l o w e s t </w>*" → "*lo w e s t </w>*".
* New tokenized corpus: [*"lo", "w", "</w>", "lo", "w", "</w>", "lo", "w", "e", "r", "</w>", "lo", "w", "e", "s", "t", "</w>"*].

Then we will repeat the third step for over the dataset.

2.3. Setting up and Training BPE

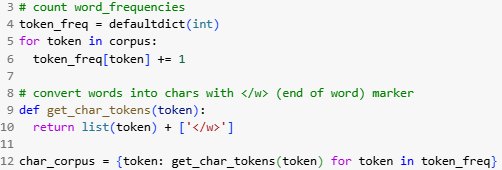
In the previous step, we applied word\_tokenize from the underthesea library to the entire dataset and got the results mentioned in the step above. Next, we will take all the words in the dataset, more specifically in the lists, and put them into a corpus so that we can proceed to build a vocab from it.

*Figure 2.8. Creating corpus from dataset.*

Then, we print out the corpus to verify.

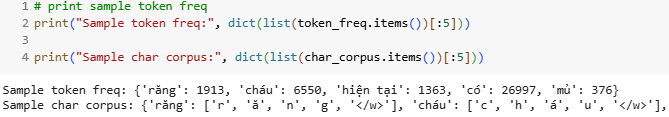
*Figure 2.9. Result of corpus.*

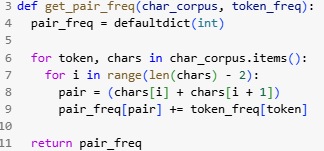
Next, as the theoretical part of the algorithm, we will find the frequency of tokens in the corpus, then save it in a dictionary so that it can be processed for the next step.



*Figure 2.10. Finding frequencies of tokens.*

And we have this result below, fully result in code file which attached with submition. There are a small crop of the result:

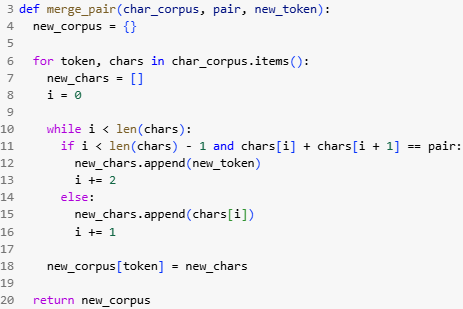
*Figure 2.11. Result of tokens’s frequencies and splitting chars.*

In the next step, we will write a function whose task is to find the pair of char tokens that appear the most. The goal of the algorithm is to combine these two characters and we will have a new vocab in which instead of just these two characters, we will have one more vocab token. This will make the vocab more diverse and learnable.

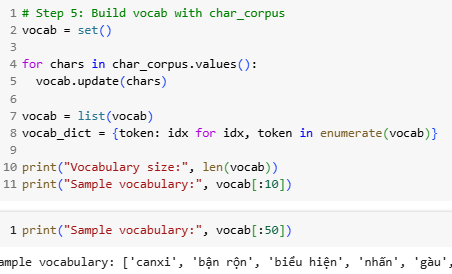
*Figure 2.12. Get pair frequency function to find most frequent pair.*

*Figure 2.13. Result of most frequent pair for first run.*

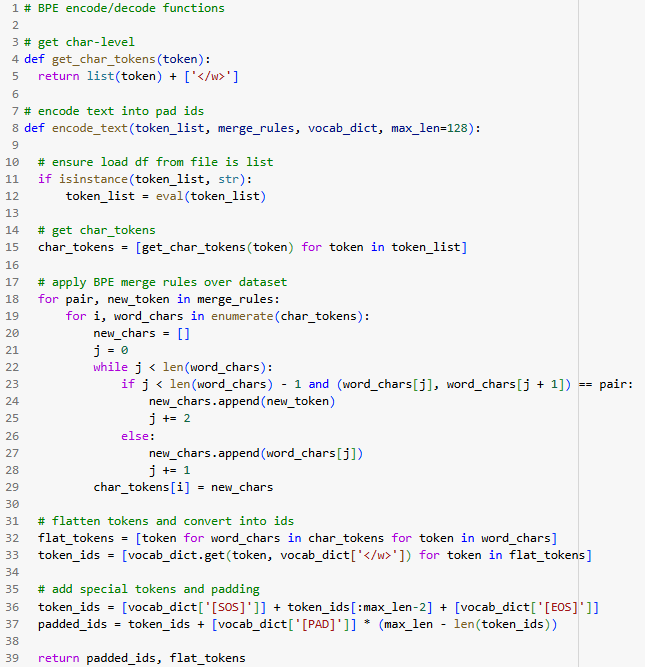
After getting the function to find the most frequently occurring character pair, we will combine them together and replace them in vocab with the function below.

*Figure 2.14. Merge pair function into vocab.*

In the last part, we just need to find the pairs that must be merged into the vocab and call them rules. Once we find the rules, we proceed to update the char\_corpus with the purpose of building the vocabulary.



*Figure 2.15. First couple of tokens in vocab.*

The most important part of bpe is done, which is how to find the rules or learn to find them. Now we will apply them to the entire dataset or encoding. We will convert the data from text or list of tokens to ids or one-dimensional matrix of each row in the dataset. From here we will use to train the model.

*Figure 2.16. Encoding dataset into ids.*

We will explain why we have the get\_char\_token function, because when we collect characters in char\_corpus, we will not collect "</w>" because it is just an end-of-word mask. And after encoding the dataset, we will have the following values, this is one of them (only take the first 10 positions).

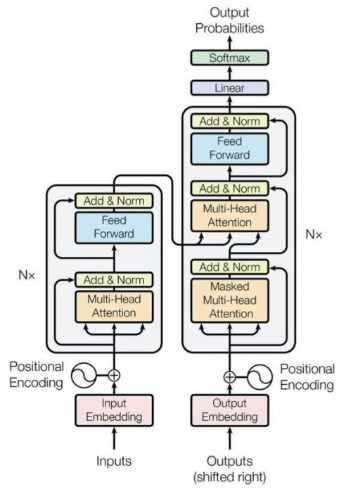
*Figure 2.17. Sample ids after encoding dataset.*

CH**APTER III – Transformer And GPT Model For Medical Q&A**

**3.1 Transformer based Encoder – Decoder architecture**

The Transformer-based encoder is a core component of the Transformer architecture, which has significantly impacted the field of Natural Language Processing (NLP) since its creation. Within NLP, the encoder's role is to convert input sequences—like words or tokens—into embeddings, which are fixed-length vectors representations that encapsulate the semantic and context of the input.

The Transformer architecture is divided into two main parts: the encoder and the decoder. The encoder processes input sequences to generate context-rich embeddings, while the decoder uses these embeddings to produce output sequences.

*Figure 3.1. General Transformer Architecture.*

**Key features of the Transformer encoder include:**

Self-Attention Mechanism: This allows the encoder to assess the relevance of different tokens within the input sequence for generating embeddings. It calculates attention scores for all token pairs, using these scores to weight and aggregate the embeddings.

Feed-Forward Neural Networks: Each encoder layer includes a feed-forward neural network that processes the outputs from the self-attention mechanism. These networks involve fully connected layers that apply non-linear transformations to enhance the input embeddings.

Layer Normalization and Residual Connections: Post each sub-layer—both self-attention and feed-forward networks—comes layer normalization, coupled with residual connections. These features are crucial for stabilizing the training process and enhancing gradient flow during backpropagation.

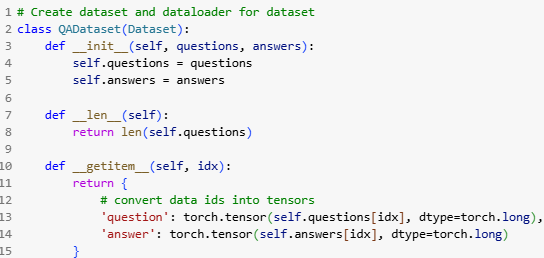
Transformers offer numerous advantages over traditional encoder models based on recurrent neural networks (RNNs), such as LSTMs or GRUs. These include more effective capturing of long-range dependencies, capability for parallel processing, and reduced susceptibility to vanishing gradients.

Overall, the Transformer encoder has been pivotal in pushing the boundaries of NLP performance, enabling advancements in machine translation, text summarization, question answering, and many other areas.

3.2 Building Transformer model from scratch and Fine – tuning T5 model

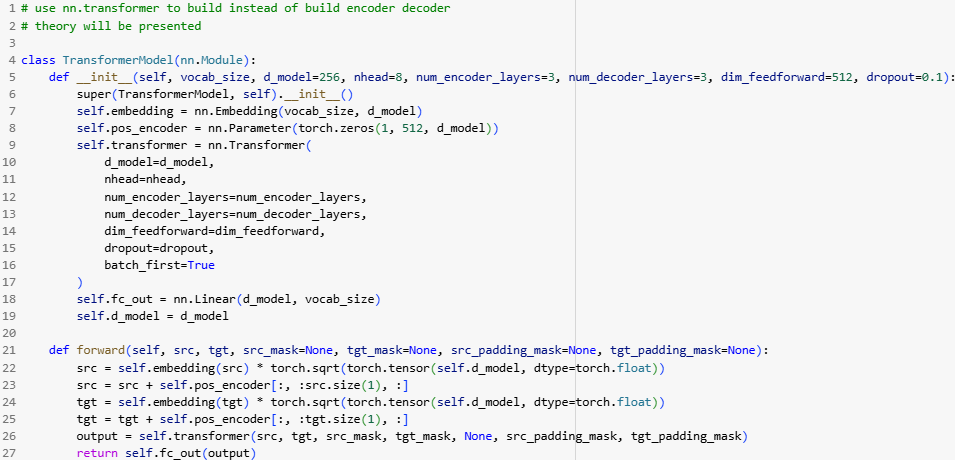
3.2.1. Transformer Building Steps

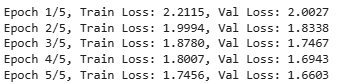
First we build a DataLoader for the model.



*Figure 3.2. DataLoader for Transformer.*

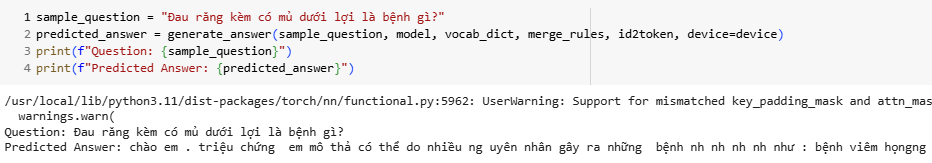
Next, we will build a Transformer model from the pre-built structure in the torch.nn library to increase the target as well as the accuracy of the results. Instead of building encoder and decoder blocks as usual.

*Figure 3.3. Transformer model class.*

Due to hardware limitations, we split the training sessions into multiple steps. We trained the model in stages on gg colab. The result below is that we trained the model for 10 epochs instead of 5. If we had more time we would improve the accuracy of the model by increasing the epoch as well as increasing the batch size.

*Figure 3.4. Loss of training and evaluating step.*

Finally below is the part where we predict the result, the result shows relatively good accuracy but then there are many overlapping characters.

**

*Figure 3.5. Result of predicted answer.*

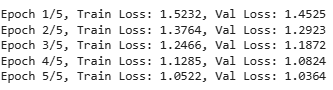
3.2.2. Fine – tuning T5 – small (A pre-trained Transformer)

Just like creating a Transformer, we will first create a T5-specific DataLoader.

*Figure 3.6. DataLoader of T5.*

In this class we can easily see the tokenizer. We will explain as follows, the tokenizer in this class is an encoding function that uses BPE merge rules like the previous model we built. For the purpose of using common training input, it can be easier to compare.

After training t5-small we get the following results.

*Figure 3.7. Result of training T5-small.*

3.3. GPT model architecture

3.3.1. Introduction of GPT-2

GPT-2 (Generative Pre-trained Transformer 2) is a generative language model developed by OpenAI in 2019.

It belongs to the transformer decoder family and is pre-trained on a large amount of text.

GPT-2 is capable of generating text fluently and in context — for example, writing paragraphs, answering questions, summarizing, translating, writing poems, etc.

GPT-2 is a transformers model pretrained on a very large corpus of English data in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts. More precisely, it was trained to guess the next word in sentences.

More precisely, inputs are sequences of continuous text of a certain length and the targets are the same sequence, shifted one token (word or piece of word) to the right. The model uses internally a mask-mechanism to make sure the predictions for the token i only uses the inputs from 1 to i but not the future tokens.

This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks. The model is best at what it was pretrained for however, which is generating texts from a prompt.

3.3.2. GPT-2 Architecture detail

**Masked Multi-Head Self-Attention**

Allows the model to learn context from only previous words (to generate the next word).

Applies attention mask to not look ahead.

Attention formula:

M: mask to hide future words

**Feedforward Layer (FFN)**

Consists of 2 linear layers:

**Residual Connection + LayerNorm**

Helps stabilize the gradient.

Apply as follows:

**GPT-2 Training Process**

**Pretraining**

GPT-2 is trained on the Language Modeling problem: predicting the next word.

Objective:

Data: tens of GB of text from the Internet (WebText)

**Fine-tuning**

GPT-2 can be fine-tuned with specific data to:

* Write specialized content
* Answer like a chatbot
* Summarize financial, medical, etc. texts
* Reinforcement Learning from Human Feedback (RLHF) can be used to control output quality.

**GPT-2 ALGORITHM AND WORKING MECHANISM**

General formula:

Training objectives:

**Masked Multi-Head Self-Attention**

Helps the model learn the relationship between previous tokens in the chain.

Create 3 matrices: Q (Query), K (Key), V (Value) from input X

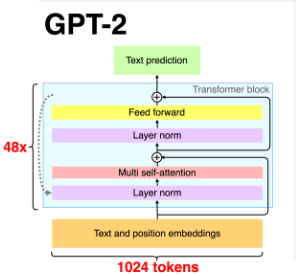
Calculate attention scores (after masking):

With multiple heads, we have:

**Feed Forward Network (FFN)**

Two linear layers that do not share weights between locations:

**Residual Connections + LayerNorm**

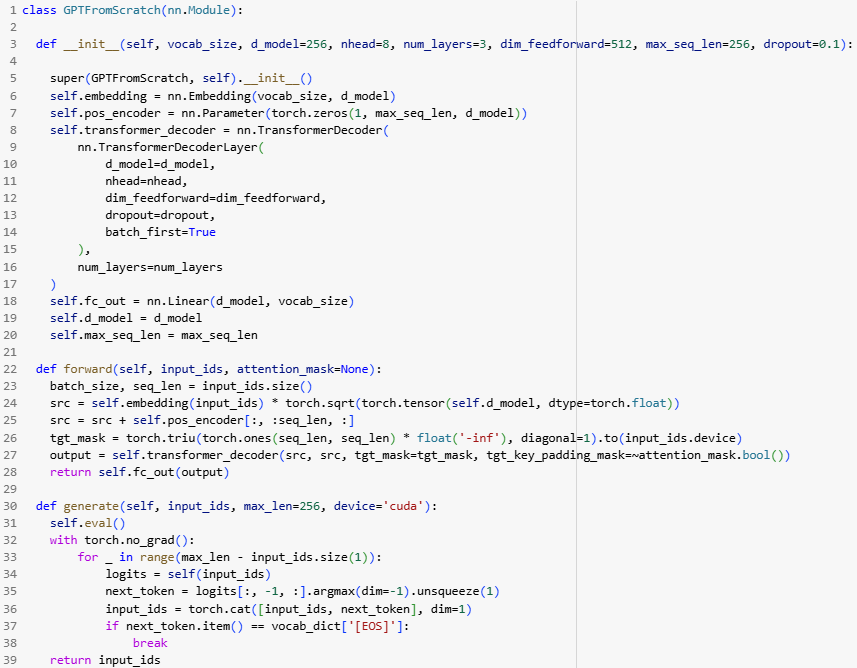
Each sublayer has:

*Figure 3.8. General architecture of GPT-2.*

3.4 Creating GPT model from scratch and Fine – tuning GPT2

Here are the ways to build a GPT model. Here are the ways to build a GPT model. We will talk about building a GPT from scratch, the fine-tuning for the GPT will be attached in the word file, as well as reported later.

*Figure 3.9. DataLoader for GPT.*

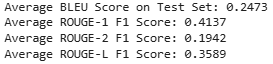
*Figure 3.10. GPT Model.*

After building the dataloader, we build the classes of the GPT model. The train and eval functions of GPT are the same as those of Transformer.

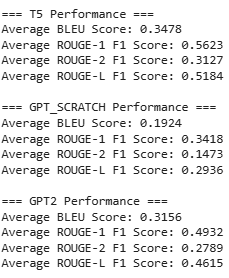
CHAPTER IV – Evaluation And Compare Models With Rouge And BLEU Score.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | BLEU | ROUGE-1 | ROUGE-2 | ROUGE-3 |
| T5 | 0.3478 | 0.5623 | 0.3127 | 0.5184 |
| GPT2 | 0.3156 | 0.4932 | 0.2789 | 0.4615 |
| Transformer | 0.2473 | 0.4137 | 0.1942 | 0.3589 |
| GPT | 0.1924 | 0.3418 | 0.1473 | 0.2936 |

*Table 2. Summary of BLEU and ROUGE.*

**

*Figure 4.1. Tranformer scores.*

**

*Figure 4.2. Others scores.*

In every significant evaluation metric, the T5 model performs noticeably better than the other four models. T5 exhibits strong capabilities in both surface-level accuracy and deeper semantic understanding, as evidenced by its BLEU score of 0.3478 and its ROUGE-1, ROUGE-2, and ROUGE-L F1 scores of 0.5623, 0.3127, and 0.5184, respectively. These high ROUGE scores show how well the model extracts the most important details from reference responses. T5 is particularly well-suited for challenging, domain-specific tasks like answering medical questions because of its pre-trained architecture and strong fine-tuning.

The GPT2 model, which was also fine-tuned on the dataset, ranks second in performance. It achieved a BLEU score of 0.3156, ROUGE-1 of 0.4932, ROUGE-2 of 0.2789, and ROUGE-L of 0.4615. While not as high as T5, these results are still strong, especially considering GPT2’s autoregressive architecture. GPT2 performs well in generating coherent and contextually appropriate responses, but it slightly trails T5 in terms of content richness and alignment with ground-truth answers.

The Transformer model, on the other hand, performs more modestly because it was constructed and trained from scratch. It performs worse than the refined models, with a BLEU score of 0.2473, ROUGE-1 of 0.4137, ROUGE-2 of 0.1942, and ROUGE-L of 0.3589. These findings highlight the difficulties of building deep learning models from the ground up, especially in specialized fields like healthcare where training resources and data availability may be constrained. The model produces partially relevant responses and demonstrates a reasonable comprehension of the task despite its lower scores.

Lastly, the GPT\_SCRATCH model, which is a GPT-style model trained from the ground up, exhibits the weakest results across all metrics. It scores only 0.1924 on BLEU, 0.3418 on ROUGE-1, 0.1473 on ROUGE-2, and 0.2936 on ROUGE-L. The difficulty of successfully training generative language models from scratch without large datasets and computational power is confirmed by the fact that these results are noticeably worse than those of the pre-trained models. The model's practical usability is limited by its inability to generate accurate and fluent responses.

REFERENCES

ENGLISH

* [1] How Transformers Work: A Detailed Exploration of Transformer Architecture. [Online]. Available: [How Transformers Work: A Detailed Exploration of Transformer Architecture | DataCamp](https://www.datacamp.com/tutorial/how-transformers-work)
* [2] Byte-Pair Encoding (BPE) in NLP. [Online]. Available: [Byte-Pair Encoding (BPE) in NLP | GeeksforGeeks](https://www.geeksforgeeks.org/byte-pair-encoding-bpe-in-nlp/)
* [3] Byte-Pair Encoding tokenization [Online]. Available: [Byte-Pair Encoding tokenization - Hugging Face LLM Course](https://huggingface.co/learn/llm-course/chapter6/5)
* [4] Byte-Pair Encoding: Subword-based tokenization algorithm. [Online]. Available: [Byte-Pair Encoding: Subword-based tokenization algorithm | Towards Data Science](https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0/)
* [5] Fine-Tuning the Pre-Trained T5-Small Model in Hugging Face for Text Summarization. [Online]. Available: [Fine-Tuning the Pre-Trained T5-Small Model in Hugging Face for Text Summarization | by Yuan An, PhD | Medium](https://medium.com/@anyuanay/fine-tuning-the-pre-trained-t5-small-model-in-hugging-face-for-text-summarization-3d48eb3c4360)