Fine Tuning QnA Chatbot Enhanced with RAG using FAISS Vector Database for Drugs Domain

Deep Learning / LC01 / 5th Semester

Disusun Oleh:

Stephen Christian Augustien	2702281241
Yoseph Oktavianus Yusanto	2702243110

Universitas Bina Nusantara
Kemanggisan
2025

Chapter 1: Introduction

1.1 Background

Access to accurate drug information is very essential for patients, healthcare students, and in public generally. However, searching through all drugs monograph and medical documents can be very complex and time-consuming. While traditional chatbots often just rely on predefined rules or limited datasets, making them inflexible and incapable of handling diverse user questions.

Retrieval-Augmented Generation is a modern approach that combines LLMs with a vector-based retrieval system to improve their accuracy, relevance, and grounding for factual information. In our project, we use FLAN-T5-base, a Seq2seq(encoder decoder) model, which is well-known for their strong instruction-following capabilities. However, models like FLAN-T5 are limited by their internal parameters and can't store all the knowledge on the domain. So, we integrate FAISS Vector Database to store document embeddings that allows the system to retrieve relevant information before generating a response.

By integrating FLAN-T5 with FAISS, the system becomes more accurate, less prone to hallucination, and increases their capabilities for grounding answers based on real documents.

1.2 Problem Statement

Access to accurate and trustworthy medication information remains a significant challenge for the general public these days. Many existing chatbot systems rely on only generative language models which may produce responses that are incomplete, outdated, or just factually incorrect, especially in the medical domain. This limitation poses serious risks, as inaccurate drug-related information can negatively impact on anyone.

Furthermore, traditional chatbot architecture often lacks mechanisms to explicitly ground their responses in authoritative medical sources. Without retrieval-based support, these systems are prone to hallucination answers and struggle to provide consistent, context-aware answers to diverse medication-related questions such as meds dosage, side effects, warnings, and usage instructions. Another challenge lies in the effective utilization of domain-specific datasets, while curated medical QA datasets such as MedInfo2019 and MedQuAD exist, they often suffer from issues such as incomplete answers or inconsistent format. This makes it difficult to build a robust and reliable medical question-answering system using a single dataset.

Therefore, there is a need for a medical chatbot system that can combine supervised fine-tuning on high-quality QA datasets with real-time retrieval of authoritative medical information. Such a system should be capable of generating accurate, context-aware, and trustworthy responses while minimizing hallucination and maintaining scalability.

1.3 Objectives

The main objective of this project include:

- Building a drug QnA chatbot with FLAN-T5-base as the generator
- Implementing a FAISS-based retrieval pipeline to provide factual grounding
- Evaluating the effectiveness of RAG in reducing wrong answers

1.4 Significance

The benefits of our projects are:

- Retrieval-Augmented Generation integration with FAISS vector database significantly improves the factual reliability for generated answers
- Use of FLAN-T5-base enables effective natural language understanding and controlled answer generation.
- Contributes on healthcare information access.
- Provides a reproducible framework for building domain-specific QA models by combining a fine-tuned generative model with a retrieval mechanism.

Chapter 2: Related Works

2.1. Seq2Seq (Encoder–Decoder) Models

FLAN-T5-base is built on the sequence to sequence architecture which includes encoder and decoder.

1. Encoder
   Processes the entire input sequence at once and operates bidirectionally while capturing full context.
2. Decoder
   Generates the output sequence one token at a time, and uses both previous tokens and the encoder's representation of the input.

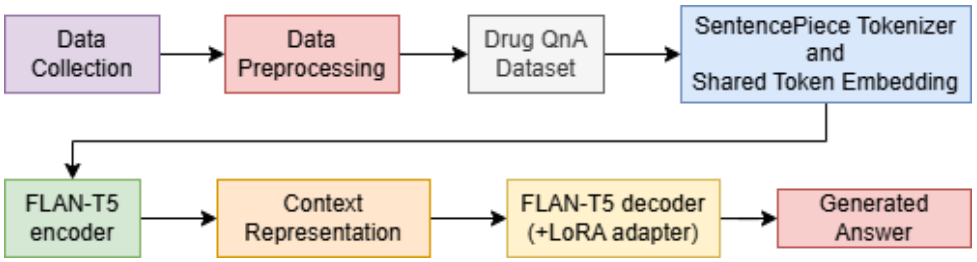
Seq2seq is ideal for RAG for their performance on question answering, highly controllable and conditionable on external context

2.2. Retrieval-Augmented Generation(RAG)

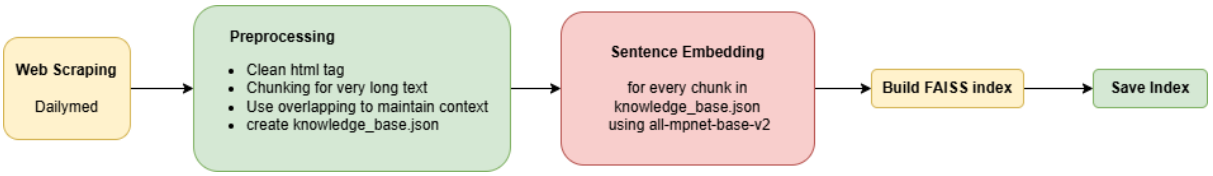
Three main components of Retrieval-Augmented Generation:

1. Embedding Model
   Converts text into dense vectors and captures semantic meaning of documents.
2. FAISS Vector Database
   Stores the embedding vectors, performs fast similarity search, and finds the most relevant document.
3. FLAN-T5 Generator
   Receives the retrieved context from FAISS, generates grounded also factual answers, and reduces hallucinations significantly.

Chapter 3: Methodology



3.1. Dataset Collection



3.1.1 DailyMed (for RAG)

Source: DailyMed Website (https://dailymed.nlm.nih.gov/dailymed/)

This study utilizes DailyMed as the primary data source for constructing the knowledge base used in the Retrieval-Augmented Generation(RAG) framework. DailyMed is an official service provided by the U.S National Library of Medicine (NLM) that offers free, reliable, and up-to-date information on marketed pharmaceutical products in the United States. The data available on DailyMed is derived from Structured Product Labeling (SPL) documents submitted by drug manufactures and approved by the U.S Food and Drug Administration(FDA).

In order to collect drug related information, this project leverages the publicly available DailyMed Application Programming Interfaces (APIs). Data retrieval is performed programmatically by submitting requests based on the drug name specified by the user.

The first API endpoint used in the data collection process is:
https://dailymed.nlm.nih.gov/dailymed/services/v1/drugname/{drug_name}/human/spls.json
This endpoint returns a JSON response containing metadata related to the queried drug, including the product title, unique identifier (setid), and the publication date of the labeling information. The setid serves as a unique reference to a specific SPL document.

Subsequently, the retrieved setid is used as a query parameter for the second API endpoint:https://dailymed.nlm.nih.gov/dailymed/services/v2/spls/{setid}.xml
This endpoint provides the complete Structured Product Labeling (SPL) document in XML format. The XML data contains detailed and authoritative drug information, such as indications, dosage and administration, contraindications, warnings, and adverse effects.

The collected SPL documents are then processed and stored as a knowledge base for the RAG system. By relying on DailyMed as the data source, the model is ensured to generate responses grounded in factual, authoritative, and trustworthy medical information, thereby reducing the risk of misinformation in drug-related question answering.

3.1.1. MedlinePlus Drug Information Dataset (for fine-tuning)

Sources:
HuggingFace MedQuAD dataset
(https://huggingface.co/datasets/lavita/MedQuAD/tree/main/data)

This is a large dataset on Medical Question Answering, that provides many document sources of medical information. This project only uses the MPlusDrugs document source that is suitable to fine tune the chatbot model. The question type on MPlusDrugs documents typically includes

- Indication
- Usage
- Precautions
- Dietary
- Severe reaction
- Contraindications, and many else

Unfortunately, for the answer column it turns out to be empty, so later on the preprocessing step, this project will use the document url that points to medlineplus website and scrape the answer from this website (https://medlineplus.gov/druginformation.html).

Purpose in the system:
This dataset is used as the fine tuning dataset because it is provided with a question and answer that later can be processed to adjust the fine tuning format of the Flan-T5 model. The answer will fine tune the model to learn the styling answer to the drug domain.

3.1.2. MedInfo2019 QA Medication Dataset (for fine-tuning)

Source:
(https://github.com/abachaa/Medication_QA_MedInfo2019/blob/master/MedInfo2019-QA-Medications.xlsx)
This dataset provides data in the format of question and answer about drug related information that come from different sources. There are many question type in this dataset, some of them are:

- Information
- Dose
- Usage
- Side effects
- Indication
- Interaction

Purpose in the system:
This dataset is used as the fine tuning dataset because it is provided with a question and answer that later can be processed to adjust the fine tuning format of the Flan-T5 model. The answer will fine tune the model to learn the styling answer to the drug domain.

3.2. Data Preprocessing

The data preprocessing stage consists of two datasets: MedInfo2019 and MedQuAD. Each dataset required different preparation steps being used for fine-tuning and evaluation.

3.2.1. MedInfo2019 Dataset Preprocessing

The MedInfo2019 dataset contains medication-related question-answer pairs in excel format. Csv file was obtained by cloning the public github repo from:
(https://github.com/abachaa/Medication_QA_MedInfo2019/)
By loading the csv file into a structured DataFrame, cleaned unnecessary whitespace, HTML characters, and formatting inconsistencies, then ensuring that each row contained a valid question and answers, the data will be clean and loaded. Then doing Standardization into Seq2seq format and Marking for final the dataset merge.

3.2.2. MedQuAD Dataset Preprocessing

the MedQuAD dataset was originally retrieved from:
(https://huggingface.co/datasets/lavita/MedQuAD/tree/main/data)
The dataset required extensive processing, especially for extraction drug-domain question answer pairs and filling the missing answers fields. firstly convert the dataset from parquet to csv format,

filtering the drug domain by only selecting rows with "document_source == MPlusDrugs", Handling missing answers:

- extracting the fields such as document_url, question, and question_type
- Fix the outdated MedlinePlus URLs
- Web scraping for the answers
- After the missing answers were handled, we exported the completed dataset and saved it as mplusdrugs_with_answer.csv then splitting the train and test dataset into 80% training and 20% testing.

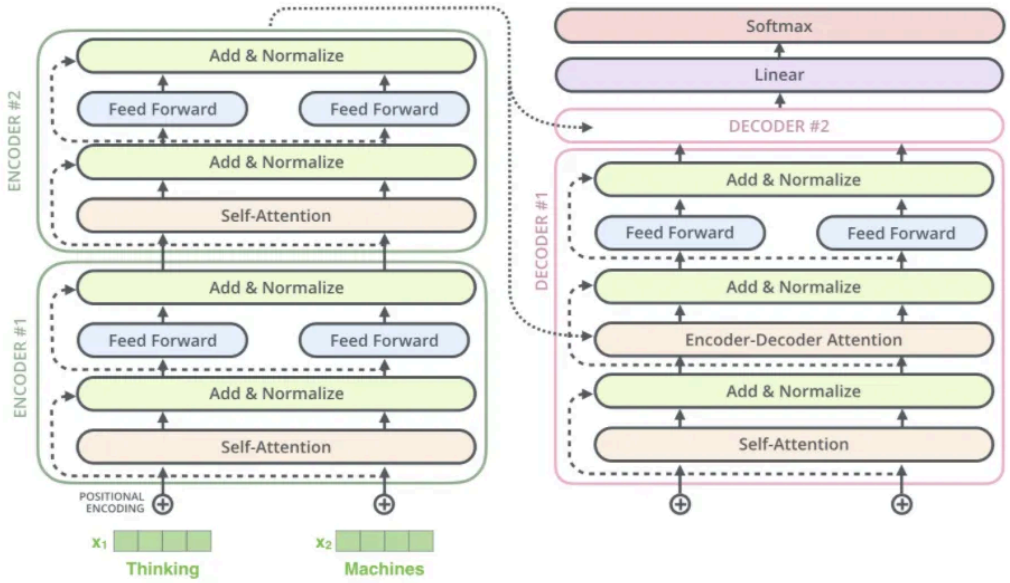
3.2.3. Final Dataset Assembly

the purpose is now to create the final training and evaluation sets:

1. Merge Sources
   both the full MedInfo2019 dataset and MedQuAD MPlusDrugs training subset which is 80% of it.
2. Remove Duplicate Questions
   Duplicate entries were detected using the exact question text matching, so only unique question-answer pairs will be trained.
3. JSON Formatting
   the final training set will be exported into train.json with the following structure will be like:

```
[
        {
         "instruction": "<question text>",
         ""output":"<answer text>"
        },
]
```

3.3 Model Architecture



FLAN-T5 is a language model based on the Transformer architecture that was built upon the Text-to-Text Transfer Transformer (T5) concept. T5 uses an encoder-decoder , where the encoder

block processes and understands the contextual representation of input text, while the decoder block generates output text autoregressively by utilizing a cross-attention mechanism on the encoder's output. One of T5 characteristics is its text-to-text approach, meaning that all NLP tasks including question answering, classification, and summarization are formulated as a mapping from input text to output text. Additionally, T5 uses relative positional encoding unlike another transformer model that uses absolute positional encoding.

FLAN-T5 retains the T5 architecture without structural changes to its layer or attention mechanism, but it enhances its performance through a process called instruction fine-tuning. In this stage, the pre-trained T5 model is further trained on FLAN dataset, which consist of various NLP tasks formatted as natural language instruction and expected outputs. The goal of this instruction fine tuning is to teach the model to better understand and execute explicit instructions provided in text form.

This is the reason for this project using FLAN-T5-base to fine tune on the available question answer dataset. Because the FLAN-T5 model is already pre-trained on the question answer dataset in the format of instruction and our dataset can be easily formatted to instruction and expected output. Also the ability of FLAN T5 base following instruction, we expect the model will obey the instruction for not answering drug questions without enough drug context from retrieval.

3.3.1 Sentencepiece Tokenizer

The FLAN-T5-base model uses SentencePiece as its tokenizer. SentencePiece is a subword-based tokenizer that undergoes a training process to determine tokens that best represent a given string. Each generated token is assigned a token ID, which is used in the shared token embedding.

The sequence probability formula:

$$P(x_1, x_2, \dots, x_n) = P(x_1)\, P(x_2 \mid x_1)\, P(x_3 \mid x_1, x_2) \cdots P(x_n \mid x_1, x_2, \dots, x_{n-1})$$

Large Language Models (LLMs) predict the next element based on the context of previous elements. However, models cannot operate directly on raw text because text is represented as strings. Therefore, a tokenizer is required to split strings into tokens, and embeddings are needed to transform tokens into numerical vectors that can be processed by neural networks such as transformers.

Dataset:

$$D = \{x^{(1)}, x^{(2)}, \dots, x^{(N)}\}$$

Each x is a string rather than a token within a dataset. If we have a dataset consisting of N string data samples, it can be mathematically modeled as shown in the equation above.

Unigram Model Formula:

$$s = \{t_1, t_2, \dots, t_k\}$$

$$P(s) = P(t_1, t_2, \ldots, t_k) = \prod_{i=1}^{k} P(t_i)$$

The unigram model calculates how frequently tokens appear within a segment s without considering the order of the tokens. A segment s consists of a number of k tokens.

Unigram LM with Latent Segmentation:

$$P(x) = \sum_{s \in S(x)} \prod_{t \in s} P(t)$$

Description:
x is a string.
P(x) is the probability of string x.
P(t) is the probability of token t, where token t is a member or part of segment s.
S(x) represents all possible segmentations of string x, the number of which can be exponential.

Therefore, during the tokenizer training process, SentencePiece considers all possible segmentations of a given string x.

Maximize Log-Likelihood Formula:

$$L = max \sum_{x \in D} log\, P(x)$$

Substituting P(x) into the equation above

$$L = max \sum_{x \in D} log \left(\sum_{s \in S(x)} \prod_{t \in s} P(t) \right)$$

This formulation is used during training to find the token parameters, specifically the token probability P(t) that best fits the data. The logarithmic function is used to ensure training stability. Without the log function, the value of P(x) would become increasingly small because P(t)<1. If the segmentation s is long and the dataset D is large, the multiplication of many P(t) values would cause P(x) to become extremely small.

Simplified Model Parameter Update Mechanism:

$$\theta_t = P(t) \;,\; \sum_{t \in V} \theta_t = 1$$

The SentencePiece model assigns an initial probability value to each unique token in the vocabulary V. The sum of all token probabilities equals one.

E-step Formula (Expected Count) for a Token t:

$$E[c(t)] = \sum_{s \in S(x)} \frac{P(s)}{\sum_{s' \in S(x)} P(s')} * c(t, s)$$

Description:
S(x) represents the segmentation of string x.
P(s) is the probability of a segmentation.
c(t,s) indicates how many times token t appears in segmentation s.
s′ represents all possible segmentations.

The expected count value is used to measure how much a particular token contributes to explaining a string.

M-step Formula:

$$Z = \sum_{t \in V} E[c(t)] \qquad \theta_{t\,new} = \frac{E[c(t)]}{Z}$$

The value Z, which represents the total expected count, is used to update the new token probability parameter P(t). After updating the probability of each token in the vocabulary, the process is repeated from the beginning until the change in likelihood becomes small or the maximum number of iterations is reached.

Inference Formula:

$$arg\,max_{s \in S(x)} \sum_{t \in s} log\, P(t)$$

P(t) is the probability of the token t learned during SentencePiece training. P(t) is not recalculated, not updated, and does not depend on context. Therefore, P(t) represents how frequently token t appears probabilistically in the training data.

During inference, the model no longer considers all possible segmentations of string x. Instead, SentencePiece selects a single best segmentation based on the learned parameters. The logarithmic form used during inference converts the multiplication of token probabilities into a summation of scores, allowing the Viterbi algorithm (dynamic programming) to operate efficiently without numerical issues.

The SentencePiece model learns to prefer longer tokens, while shorter tokens that rarely stand alone are assigned lower probabilities. Tokens that represent strings most efficiently are assigned higher probabilities.

3.3.2 Shared Token Embedding

FLAN-T5-Base has a vocabulary size of approximately 32,000 tokens with an embedding dimension of 768. It is referred to as shared token embedding because the same embedding matrix is used for encoder input embeddings, decoder input embeddings, and the output softmax projection. In T5, absolute positional embeddings are not used; instead, relative position bias is applied as a replacement for positional embeddings.

After SentencePiece tokenization, a sequence of n tokens is obtained

$$x = \{x_1, x_2, x_3, \dots, x_n\}$$

Each token is mapped to an embedding vector

$$e_i = E[x_i],\ E \in R^{V \times d}$$

The embedding matrix consists of a number of rows equal to the number of unique tokens or vocabulary size in the dataset, and a number of columns equal to the embedding dimension of 768, as used in FLAN-T5-Base.

As a result, the input to the transformer is obtained as follows:

$$X = [e_1, e_2, e_3, \dots, e_n]\ ,\ X \in R^{n \times d}$$

The input vector consists of nnn embeddings, where each embedding has a dimensionality of 768.

3.3.3 Encoder : Transformer Encoder Stack

Big concept:

$$Encoder(X) = H \quad , \quad H = (h_1, h_2, h_3, \dots, h_n)\ ,\ h_i \in R^d$$

Description:
H is the hidden state.
h_i is the representation vector of the i-th token after incorporating the full input context.

Each encoder layer consists of two main blocks: Self-Attention and a Feed Forward Network (FFN).

1. Self-Attention

For each token, the following three matrices are generated:

$Q = XW_Q,\ K = XW_K,\ V = XW_V$

For each token in the input sequence within the same attention head, the same projection matrices W_Q , W_K , dan W_V are used. Different heads use different weight matrices.

Self Attention Formula:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}} + B\right)V\ ,\ B \in R^{H \times B'}$$

Description:
B is a predefined bias matrix or bias table.
H is the number of attention heads.
B’ is the number of buckets.

The multiplication between matrix Q and the transpose of matrix K produces relationships between each token and all other tokens, including itself. The multiplication of rows of matrix Q and columns of the transposed matrix K is referred to as the dot product. This dot product is used to measure the similarity between the query of one token and the key of another token. As a result, multiplying matrix Q by the transpose of matrix K produces a matrix of unscaled dot-product similarities for all possible query–key combinations.

Next, the dot-product similarity matrix is scaled by dividing it by the square root of the dimensionality of the columns of matrix K, which corresponds to the embedding dimension of a token. The result of this scaling operation is the scaled dot-product similarity matrix.

Breakdown Relative Bias:

T5 does not use absolute positional embeddings, instead it applies relative position bias in each self-attention mechanism. The intuition behind this approach is to represent how far one token is from another token, rather than encoding the absolute position of a token as done in absolute positional embeddings. In this component, embeddings gain contextual meaning through relative positioning.

$$r = j - i \Rightarrow bucket(r) \in \{0, 1, \dots, B - 1\}$$

Description:

r is relative distance

B is the number of buckets.

The bucket function maps a relative distance r to an integer index. Buckets are not model parameters but rather rules for grouping relative distances.

$$score\, i,j = \frac{q_i \cdot k_j}{\sqrt{d_k}} + b_{h,k},\ k = bucket(r)$$

Description:

q_i is the i-th row of matrix Q, representing the query vector of a token.

k_j is the j-th column of the transposed matrix K, representing the key vector of a token.

$b_{h,k}$ is a scalar bias parameter, where b is a learned model parameter whose value is updated during training.

Steps to Obtain the Bias Parameter ($b_{h,k}$) :

1. The model already contains a bias table B.
2. For each token pair(i,j) the relative distance is computed as r = j - i and k = bucket(r)
3. The bias value is then retrieved from the bias table based on the attention head index and index k.

Next, for each row of the scaled dot-product similarity matrix that has already been augmented with the bias score, the values are passed through the softmax function so that the sum of the dot-product values in each row equals one. The softmax operation can be interpreted as summarizing the relationships between tokens into proportional weights.

The final step multiplies the softmax-normalized scaled dot-product similarity matrix with the Value matrix. This multiplication produces the self-attention score for each input token.

Multi-head attention:

$$MultiHead(X) = Concat(head_1, head_2, head_3, \dots, head_n)Wo$$

Each $head_i$ is an independent self-attention matrix resulting from the self-attention computation for a single head. These head matrices are concatenated to form multi-head attention, enabling the model to learn multiple types of contextual relationships. The matrix Wo purpose is to combine information across different attention heads. Wo is a learned output projection that linearly combines the concatenated head outputs into the model embedding space.

2. Feed Forward Network (FFN)

Gated Feed Forward Network (FFN) in T5:

$$FFN(x) = \left(W_2\sigma\left(W_1x\right)\right) \circ \left(W_3x\right),\ x \in R^{n\times d}$$

Description:

The activation function can be ReLU or GELU.

Element-wise multiplication is applied.

x represents the token representation after attention, residual connection, and layer normalization, where the number of rows corresponds to the number of tokens and the number of columns is 768.

Residual Connection + Layer Normalization:

$$H = X + FFN(LayerNorm(X))$$

The residual connection adds the original input X to the output of the feed-forward network, allowing the layer to learn a residual transformation. This facilitates gradient flow, stabilizes training in deep architectures, and enables the model to preserve identity mappings when needed.

Output Encoder:

$$H = (h_1, h_2, h_3, \dots, h_n)$$

3.3.4 Decoder: Transformer Decoder Stack

The decoder consists of three sub-layers.

1. Masked Self Attention

$$MaskedAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}} + B + M\right)V$$

The matrix M is used to prevent the decoder from accessing similarity scores of tokens that are yet to be predicted, effectively preventing data leakage. The structure of matrix M is as follows:

$$M = \begin{pmatrix} 0 & -\infty & -\infty & \cdots & -\infty \\ 0 & 0 & -\infty & \cdots & -\infty \\ 0 & 0 & 0 & \cdots & -\infty \\ \vdots & \vdots & \vdots & \ddots & -\infty \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

A value of zero keeps the similarity score unchanged because adding zero does not alter the value, whereas a value of negative infinity forces the similarity score to negative infinity. When passed through the softmax function, values of negative infinity result in zero probability. A zero probability

indicates that the corresponding token is excluded from consideration when determining the next token prediction.

residual connection:

$$Y' = Y + MaskedAttention(LayerNorm(Y))$$

2. Encoder Decoder Cross Attention

$$CrossAttention(Y', H) = softmax\left(\frac{QK^T}{\sqrt{d_k}} + B\right)V$$

Cross-attention in this component follows the same computation as self-attention in the encoder. However, the Query matrix is derived from the decoder output multiplied by the matrix W_Q while the Key and Value matrices are derived from the encoder output multiplied by W_K dan W_V, respectively.

$Q = Y'W_Q,\ K = HW_k,\ V = HW_v$

Description:
Y' is the decoder output.
H is the encoder output.

residual connection:

$$Y'' = Y' + CrossAttention(LayerNorm(Y'))$$

3. Feed Forward Network

$$FFN(x) = \left(W_2\sigma\left(W_1x\right)\right) \circ \left(W_3x\right),\ x \in R^{n\times d}$$

residual connection:

$$Y''' = Y'' + FFN(LayerNorm(Y''))$$

Output:projection and softmax
hidden state decoder:

$$H^{dec} = (h_1, h_2, h_3, \dots, h_T), H^{dec} \in R^{T\times d}$$

Description:
T is the length of the output sequence or the number of generated tokens (teacher forcing).
d is the model dimension (768).

logits:

$$Z = H^{dec}W_{out} + b$$

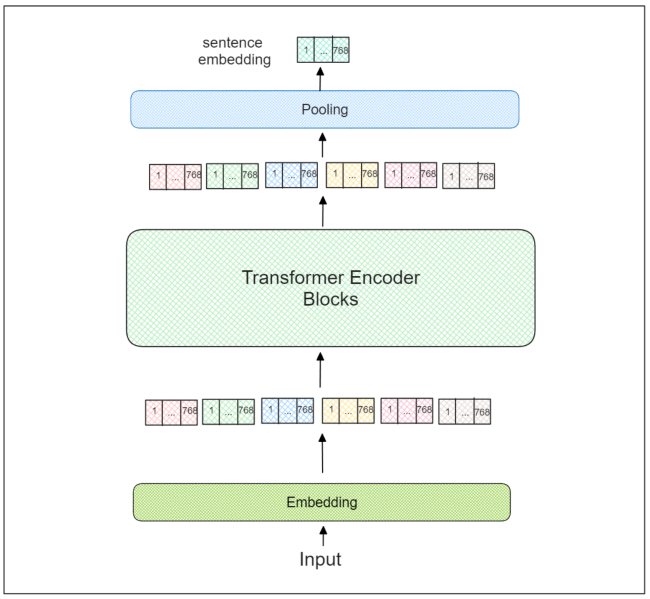
The values in the logits are passed through the softmax function to obtain the probability of each token.

Training Loss Function
Log-likelihood:

$$L(\theta) = \sum_{(x,y)} \sum_{t=1}^{m} log\, P(yt \mid y < t,\, x; \theta)$$

θ represents the model parameters, x denotes the input text sequence and y is the target output sequence. At each time step t, the model predicts the probability of the target token yt conditioned on all previously generated target tokens $y < t$ as well as the input representation x produced by the encoder. This probability is obtained from the softmax output of the Transformer decoder. By summing the log-probabilities over all tokens in the target sequence, the loss function encourages the model to generate output sequences that are most consistent with the training data. In practice, the log-likelihood is typically negated, resulting in a negative log-likelihood or cross-entropy loss, which is minimized during training using a teacher forcing scheme.

3.3.5 all-mpnet-base-v2



The knowledge base scraped from the DailyMed website must be embedded to produce vector representations that can later be retrieved using queries through a FAISS vector database. Traditional embedding models such as Word2Vec are limited to capturing context at the word level and are not well suited for representing long-form textual content. In contrast, MPNet (Masked and Permuted Language Model) is a sentence embedding model that employs a Transformer-based architecture as its backbone.

MPNet combines the strengths of Masked Language Modeling (MLM), as used in BERT, and Permuted Language Modeling (PLM), as introduced in XLNet, while addressing the limitations of both approaches. Although BERT is well known for its bidirectional contextual modeling, allowing a token to be predicted based on both its left and right context. It relies on MLM, where randomly masked tokens are predicted independently. This independence assumption prevents the model from leveraging dependencies among masked tokens, even though predicting one token could provide useful information for predicting subsequent masked tokens. On the other hand, XLNet's autoregressive, permutation-based training enables richer contextual modeling but sacrifices absolute positional information, which is essential for understanding sentence structure and word order.

MPNet overcomes these limitations by integrating PLM with a two-stream self-attention mechanism. This design allows the model to capture dependencies between predicted tokens while simultaneously preserving access to the full positional information of the input sequence. Trained on

approximately one billion text pairs, the all-mpnet-base-v2 model maps variable-length textual inputs into a fixed 768-dimensional dense vector space. These embeddings form the core data representation within the FAISS vector database, enabling efficient semantic retrieval and clustering through vector similarity measures such as cosine similarity.

The knowledge base data that consists of drug names, categories, section titles, and detailed content text, comprises relatively long textual sequences. Therefore, MPNet, with its fixed 768-dimensional embedding space and strong sentence-level semantic modeling capabilities, is particularly well suited for embedding this type of data.

3.4 Training Setup

Model Configuration:

- Base model: Google FLAN-T5-base
- Architecture: seq2seq (Encoder-Decoder)

Device: GPU RTX 4060 vram 8GB (CUDA)

LoRA Configuration:

- LoRA Rank : 8
- LoRA Scaling Factor : 32
- LoRA Dropout: 0.05

Data Split:

- Training: 80% (3449 samples)
- Testing: 20% (699 samples)

Training Configuration:

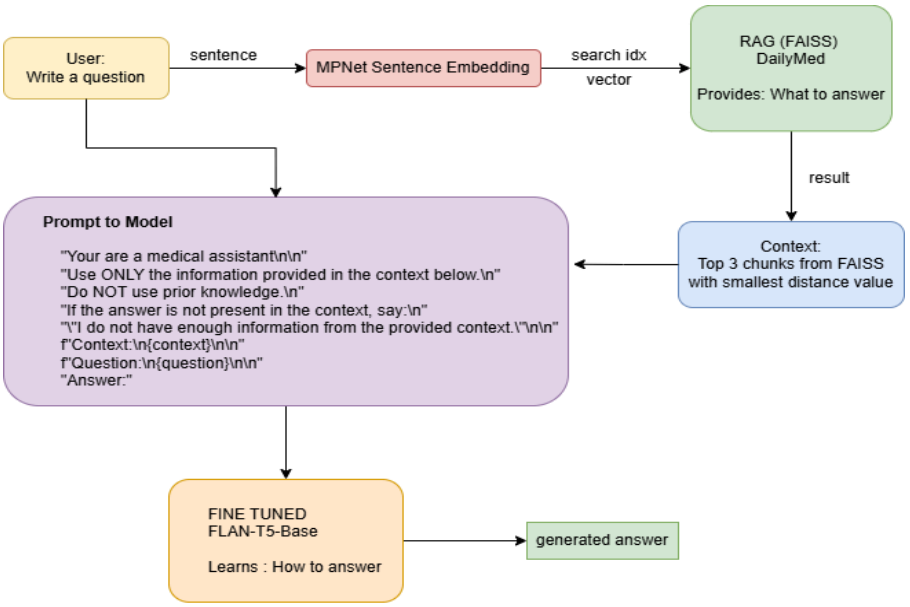
- Epoch: 5
- Batch size: 8
- Gradient Accumulation Step: 2
- Per device train batch size: 4
- Learning rate: 0.0001
- Max length: 256
- Warmup ratio: 0.03

3.5 Evaluation Metrics

The performance of the fine-tuned Flan-T5-base model is evaluated using ROUGE-1, ROUGE-2, ROUGE-L, BLEU, and F-score, which are standard metrics for assessing text generation quality by comparing generated outputs with reference texts. ROUGE-1 and ROUGE-2 measure unigram and bigram overlap, respectively, indicating content coverage and local word order accuracy, while ROUGE-L evaluates the longest common subsequence to capture sentence-level structural similarity. BLEU focuses on n-gram precision, measuring how accurately the generated text matches the reference and penalizing irrelevant or hallucinated content. The F-score provides a balanced measure of precision and recall, reflecting both the completeness and correctness of the generated outputs.

The quality of the FAISS-based retriever is evaluated using Recall@K, which measures whether relevant documents are retrieved within the top K results. A higher Recall@K indicates more effective retrieval, ensuring that relevant contextual information is available to support the generation process.

Chapter 4: Implementation & Result



4.1 Implementation

After building the FAISS vector database and fine-tuning the FLAN-T5 Base model, the trained components were integrated into the chatbot application. Due to the use of Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA), the fine-tuning process does not modify or overwrite the original pretrained model parameters. Instead, only a small set of task-specific adapter weights is learned and stored.

As a result, the saved model directory contains only the LoRA adapter files, including the adapter weights and configuration, along with the tokenizer files. The original FLAN-T5 Base model parameters are not duplicated in order to reduce storage requirements and improve deployment efficiency.

During the implementation phase, the chatbot model is loaded using a two-step process. First, the pretrained FLAN-T5 Base model is loaded either from the local cache or a local directory. Second, the LoRA adapter weights obtained from the fine-tuning stage are attached to the base model to reconstruct the complete fine-tuned model. This approach enables the chatbot to retain the general language understanding capabilities of FLAN-T5 while incorporating domain-specific knowledge related to drug question answering.

4.2 Fine tuning Flan-T5-base results

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Bleu	F1
1	4.030700	3.559692	0.110115	0.015623	0.099892	0.008779	0.041341
2	3.735000	3.256807	0.158803	0.053070	0.147189	0.038538	0.060716
3	3.579000	3.121470	0.252317	0.144494	0.234268	0.097129	0.137150
4	3.508200	3.060488	0.302213	0.193377	0.277246	0.130816	0.181346
5	3.456300	3.042266	0.304649	0.194534	0.278225	0.133254	0.182185

The FLAN-T5 Base model was successfully fine-tuned on an instruction-output dataset using a parameter-efficient-fine-tuning (PEFT) approach with LoRA, where only approximately 0.36% of the model parameters (884.746 out of 248 million) were updated during training. The model was trained for five epochs using 3.449 training samples and evaluated on 699 evaluation samples, with performance assessed at the end of each epoch.
The training results show a steady decrease in training loss from 4.03 in the first epoch to 3.46 in the final epoch, alongside a reduction in validation loss from 3.56 to 3.04, indicating that the model progressively learned the instruction-following patterns without exhibiting significant overfitting. In addition, text generation quality metrics demonstrated consistent improvement, with ROUGE-1 increasing from 0.11 to 0.30, ROUGE-2 from 0.02 to 0.19, and ROUGE-L from 0.10 to 0.28, reflecting better content alignment and structural similarity between generated responses and reference answers.

Despite these improvements, the obtained BLEU score and F1-score remain relatively low, reaching 0.133 and 0.182, respectively, in the final epoch. This outcome is expected in the context of open-ended generative question answering, particularly in the drug information domain, where a single question can admit multiple valid answers that differ lexically while remaining semantically equivalent. BLEU and token-level F1 metrics rely heavily on exact word overlap and token order, making them less effective at capturing semantic similarity in free-form text generation. Consequently, lower BLEU and F1 scores do not necessarily indicate poor model performance but rather highlight the limitations of token-matching-based evaluation metrics for generative QA tasks. In this setting, the consistent improvement in ROUGE metrics, together with qualitative inspection of generated responses, provides a more representative assessment of the model's ability to produce relevant, informative, and contextually appropriate answers for a drug information chatbot.

4.3 Result of FAISS retriever evaluation:

- Recall@1 : 0.625
  Meaning:
  From 100 query to the FAISS vector database, 62- 63 query is successfully found with a minimum of 1 correct chunk in the top 1 chunk retrieved from FAISS vector database.
- Recall@3: 0.675
  Meaning:
  From 100 query to the FAISS vector database, 67- 68 query is successfully found with a minimum of 1 correct chunk in the top 3 chunk retrieved from FAISS vector database.
- Recall@5: 0.675
  Meaning:
  From 100 query to the FAISS vector database, 67- 68 query is successfully found with a minimum of 1 correct chunk in the top 3 chunk retrieved from FAISS vector database.

Chapter 5: Discussion & Limitation

The experimental results demonstrate that integrating FAISS-based retrieval with FLAN-T5-base is significant in improving the reliability of medication-related question answering. The RAG-enhanced system consistently generated answers that were grounded in authoritative MedlinePlus drug information, which effectively reduced the occurrence of unsupported or fabricated content commonly observed in standalone generative models. Then, the Seq2Seq encoder-decoder

architecture of FLAN-T5 enabled the model to fully process the retrieved documents before generating responses, allowing better contextual understanding and alignment between the question and the supporting information. This observation is consistent with prior chatbot studies that emphasize the importance of contextual comprehension in NLP-based conversational systems. In addition, stratified evaluation across different question_type categories like dosage, side effects, and warnings, showed stable performance, indicating that the model generalized well across a wide range of medication-related inquiries.

Limitation:

- Long or complex drug interaction questions occasionally produced partial answers.
- The system is not designed to replace professional medical consultation.
- Performance depends heavily on the quality of retrieved documents.

Nevertheless, compared to traditional chatbot architectures discussed in, the proposed system demonstrates a more flexible and knowledge-grounded approach.

Chapter 6 : Conclusion & Future Work

This study presents a drug question-answering chatbot that integrates FLAN-T5-base, a Seq2Seq transformer model, with Retrieval-Augmented Generation using FAISS Vector Database. The combination of supervised fine-tuning on medication QA data and document retrieval from MedlinePlus enables the system to generate more accurate, relevant, and trustworthy responses compared to traditional chatbot approaches.

The experimental results confirm that RAG is an effective strategy for mitigating hallucination and improving factual consistency in medical chatbot systems. Future work may explore larger language models, improved evaluation metrics, and real-time updating of medical knowledge sources.

References

T. L. M. Suryanto, A. P. Wibawa, H. Hariyono, and A. Nafalski, “Evolving Conversations: A Review of Chatbots and Implications in Natural Language Processing for Cultural Heritage Ecosystems,” International Journal of Robotics and Control Systems, vol. 3, no. 4, pp. 955–1006, Dec. 2023, doi: 10.31763/ijrcs.v3i4.1195.

I. D. Raharjo and E. R. Subhiyakto, “Implementing Long Short Term Memory (LSTM) in Chatbots for Multi Usaha Raya,” *Advances in Sustainable Science, Engineering and Technology*, vol. 6, no. 4, p. 02404018, Oct. 2024, doi: 10.26877/asset.v6i4.934.

S. Hawanti and K. M. Zubaydulloevna, “AI chatbot-based learning: alleviating students’ anxiety in English writing classroom,” *Bulletin of Society for Informatics Theory and Application*, vol. 7, no. 2, pp. 182–192, Dec. 2023, doi: 10.31763/BUSINTA.V7I2.659.

P. D. Larasati, A. Irawan, S. Anwar, M. F. Mulya, M. A. Dewi, and I. Nurfatima, “Chatbot helpdesk design for digital customer service,” *Applied Engineering and Technology*,

Accessed: Dec. 20, 2024. [Online]. Available: https://pubs2.ascee.org/index.php/aet/article/view/684.

Dataset:

A. Ben Abacha, Y. Mrabet, M. Sharp, T. Goodwin, S. E. Shooshan, and D. Demner-Fushman, "Bridging the gap between consumers' medication questions and trusted answers," *MEDINFO 2019*, 2019.

A. Ben Abacha and D. Demner-Fushman, "A question-entailment approach to question answering," *BMC Bioinformatics*, vol. 20, no. 1, pp. 511:1-511:23, 2019. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-019-3119-4

Appendix

1. Team Contributions

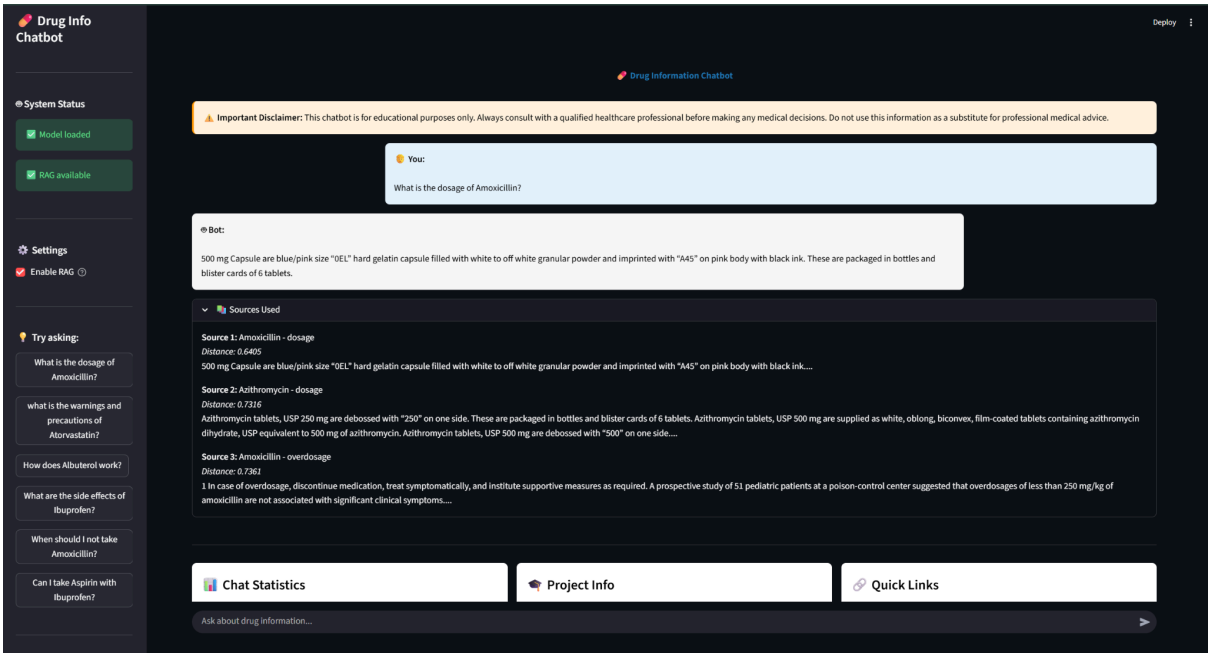
Task	**PIC**	**Task Description**
Find Dataset	Stephen, Yoseph	Yoseph: searching for drug domain dataset from various sources like kaggle, huggingface, github, and mendeley. Also doing the web scraping Stephen: searching for drug domain dataset from various sources
EDA	Yoseph	Perform EDA
Preprocessing	Yoseph	Perform Preprocessing
Model Creation	Yoseph	Perform fine tuning
Model Evaluation	Yoseph	Perform evaluation on fine tuned model and quality of the retriever augmented generation
Create App	Yoseph	Integrate chatbot consisting of model and RAG into streamlit that can be used by users
Final Report	Stephen, Yoseph	Yoseph: write chapter 3 (methodology) and chapter 4

		(result & implementation) Stephen: write chapter 1 (introduction), 2 (related work), 5 (discussion & limitation), and 6 (conclusion & future work)
Presentation	Stephen, Yoseph	prepare presentation slide

2. Supporting Documents

No	Document	Link
1.	Github Repo	https://github.com/yosephyusanto/FinalProject-DeepLearning
3.	Demo Video	https://youtu.be/evrFux7Xfxk
4.	Presentation Slide	https://www.canva.com/design/DAG7ehGveow/aJXaijPYkePBHQl7ECZAHA/edit?utm_content=DAG7ehGveow&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

3. Screenshots and Code Snippets



Training

```python
def train_model(model, tokenizer, train_dataset, test_dataset, model_type):
  print("Starting Training...")

  # Training arguments
  training_args = Seq2SeqTrainingArguments(
    output_dir=str(OUTPUT_DIR),
    num_train_epochs=TRAINING_CONFIG['num_epochs'],
    per_device_train_batch_size=TRAINING_CONFIG['batch_size'],
    per_device_eval_batch_size=TRAINING_CONFIG['per_device_eval_batch_size'],
    gradient_accumulation_steps=TRAINING_CONFIG['gradient_accumulation_steps'],
    learning_rate=TRAINING_CONFIG['learning_rate'],

    # warmup_steps=TRAINING_CONFIG['warmup_steps'],
    warmup_ratio=TRAINING_CONFIG['warmup_ratio'],
    weight_decay=0.01,
    label_smoothing_factor=0.05, # 0.1 , 0.5

    logging_steps=100, # increase to reduce logging frequency : 10, 100
    eval_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=2,
    load_best_model_at_end=True,
    metric_for_best_model="rouge1", # if using rouge1 then greater is better True, if using eval_loss then False

    greater_is_better=True, # because we use rouge1 and the higher the better
    fp16=False, # Disable mixed precision to avoid errors
    bf16=True, # Use bfloat16 if supported (i used rtx 4060 so i enable this)
    predict_with_generate=True,  # important for seq2seq to generate during evaluation, avoid list of list of float (logits) to decode()
    generation_max_length=TRAINING_CONFIG['max_length'],
    optim="adamw_torch",
    report_to="none",  # Disable wandb, tensorboard
  )

  # Data collator
  data_collator = DataCollatorForSeq2Seq(
    tokenizer=tokenizer,
    model=model,
    padding=True
  )

  # Initialize trainer
  trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    data_collator=data_collator,
    compute_metrics = lambda eval_pred : compute_metrics(eval_pred, tokenizer)
  )

  # Train
  print("Training started...")
  trainer.train()
  print("Training completed.")
```