

Fine-Tuning Pipeline Tutorial Documentation for FLAN-T5 Family

Core Concepts: LLMs, FLAN-T5, and Fine-Tuning

What is LLM?

A Large Language Model (LLM) is an artificial intelligence (AI) model that has been trained on massive volumes of text data. The purpose of these models is to comprehend, produce, and work with human language. They accomplish a wide range of language-based activities, including composing essays, translating languages, answering questions, and summarising texts, by acquiring patterns, grammar, facts, and reasoning skills from the material they are educated in.

Deep learning architectures, most frequently the Transformer architecture, are used to build LLMs. By focussing on distinct elements of the input sequence, this design enables them to process sequential material, such as text, and comprehend context and word connections even when they are widely apart.

What is Flan-T5?

FLAN-T5 is a family of LLMs created by Google. It represents an improvement over the original T5 (Text-to-Text Transfer Transformer) model. Its comprehensive fine-tuning on a vast collection of instruction-following datasets is the main uniqueness of FLAN-T5. This indicates that FLAN-T5 has been trained to comprehend and react to tasks that are explained in instructions in natural language.

FLAN-T5 is very adaptable and can be tailored to many downstream jobs with little further training because it is built to obey instructions. FLAN-T5 is capable of the following tasks, for example:

1. Summarisation: Condensing long texts into shorter summaries.
2. Translation: Converting text from one language to another.
3. Question Answering: Providing answers to questions based on given context.
4. Text Generation: Creating new text, such as stories, poems, or code.
5. Classification: Categorising text (sentiment analysis).

What is Fine-Tuning?

A key step in transfer learning is fine-tuning, which involves further training a pre-trained model (such as FLAN-T5) on a smaller, task-specific dataset. The objective is to master a specific task or subject by adapting the general information acquired during the first, extensive training phase.

Consider it this way: an LLM who has received pre-training is comparable to a highly educated person with extensive knowledge. Giving the person specialised training for a particular task is analogous to fine tuning. Although they already possess the fundamental abilities, they are now learning the subtleties and particular demands of their new position.

Why fine-tune?

1. Specialisation to make the model perform exceptionally well on a specific task (medical text analysis, legal document summarization).
2. Improved accuracy to achieve higher accuracy on our target task compared to using the base pre-trained model.
3. Obtain efficiency than training a large model from scratch, as it leverages the existing learned knowledge.

Tutorial Introduction

This tutorial outlines a detailed, step-by-step process for fine-tuning FLAN-T5 models. FLAN-T5, an extension of the T5 (Text-to-Text Transfer Transformer) model, is fine-tuned on a collection of instruction-following datasets. This makes it highly effective for a wide range of natural language understanding and generation tasks when further adapted to specific domains or tasks.

The FLAN-T5 family is a series of instruction-tuned models developed by Google, based on the T5 (Text-to-Text Transfer Transformer) architecture. These models are optimised to follow instructions across a wide range of tasks.

In this tutorial, FLAN-T5 is fine-tuned on the MedDialog (English) dataset, which consists of patient–doctor dialogues. The objective is to train a telehealth assistant capable of generating short, clear, and helpful responses to patient queries.

Instead of fully fine-tuning all parameters of FLAN-T5 (which is computationally expensive), we use LoRA (Low-Rank Adaptation), a Parameter-Efficient Fine-Tuning (PEFT) method. LoRA enables training only a small fraction of parameters while keeping the base model frozen, thus reducing training time and hardware requirements.

Pipeline Overview

The fine-tuning pipeline involves several key stages:

Step 1. Environment Setup: Installing necessary libraries (Hugging Face Transformers, PEFT, PyTorch) and configuring the hardware (GPU).

Step 2. Data Preparation: Gathering, cleaning, and formatting the dataset for the specific task. This involves creating input-output pairs.

Step 3. Model Selection and Loading: Choose a suitable pre-trained LLM architecture (FLAN-T5-base). Loading the pre-trained FLAN-T5 model and tokenizer. Construct tokenizer initialisation and Preprocessing.

Step 4. Fine-Tuning Strategy: LoRA Configuration to define the LoRA parameters, such as rank (r), alpha (lora_alpha), dropout (lora_dropout), and target modules (query, key, value, output layers).

Step 5. Training Configuration: Setting up hyperparameters for the training process and training arguments, including learning rate, batch size, number of epochs, weight decay, and logging steps.

Step 6. Model Training: Executing the fine-tuning process using the configured LoRA adapters and training arguments.

Step 7. Evaluation: Assessing the performance of the pretrained model and fine-tuned model using evaluation metrics of ROUGE and BERTScore.

Step 8. Deployment: Preparing the model for model inference.

Step 1. Environment Setup

Before proceeding, ensure we have the following:

- **Python:** Version 3.10.
- **GPU Model:** NVIDIA Tesla T4 GPU
- **PyTorch** or **TensorFlow:** FLAN-T5 models are available for both popular deep learning frameworks.
- **Hugging Face Transformers:** This is the go-to library for accessing and working with pre-trained models like FLAN-T5. It provides easy-to-use APIs for downloading, loading, and training models.
- **Datasets:** A library from Hugging Face that simplifies loading, processing, and managing large datasets efficiently.
- **Accelerate:** A library from Hugging Face that helps streamline distributed training (training across multiple GPUs or machines) and makes it easier to use mixed-precision training for faster and more memory-efficient training.

- **Hardware:** A GPU (Graphics Processing Unit) is highly recommended. Training LLMs is computationally intensive, and GPUs can accelerate this process significantly compared to CPUs.
- **Tokenizers:** a tokenization library for converting text into a suitable format for the use case
- **Evaluate:** This library provides a wide range of metrics for model evaluation, ensuring that the fine-tuned model meets the desired performance standard

Installation of the libraries

The installation can be performed as follows, using pip, the Python package manager from Jupyter notebook.

```
pip install transformers datasets evaluate peft accelerate bitsandbytes rouge_score bert_score torch
```

Import libraries

After the installation of all the libraries, we can import

```
import torch
import pandas as pd
import matplotlib.pyplot as plt

from datasets import Dataset, DatasetDict
from sklearn.model_selection import train_test_split
from transformers import T5Tokenizer
from transformers import T5ForConditionalGeneration, BitsAndBytesConfig
from peft import get_peft_model, LoraConfig, TaskType
from transformers import Seq2SeqTrainingArguments, Seq2SeqTrainer
from evaluate import load
from datasets import Dataset
```

A well-configured environment ensures that all the necessary tools and libraries are available and compatible, leading to a smoother and more efficient training process. Hugging Face's libraries abstract complex functionalities, making LLM fine-tuning accessible.

Step 2: Data Preparation

The quality and format of our data are crucial for successful fine-tuning. FLAN-T5 is particularly effective with instruction-formatted data.

Dataset Description

The MedDialog-English dataset contains thousands of medical dialogues between patients and doctors. Each conversation includes multiple turns, but in this pipeline we focus on single-turn exchanges: one patient's utterance followed by one doctor's response.

- Source Files: english-train.json, english-dev.json, english-test.json

- Structure: JSON list of dialogues with utterances fields

Data Cleaning and Filtering

1. Only dialogues with at least one patient and one doctor utterance are kept.
2. Patient utterances are cleaned by removing the “patient:” prefix.
3. Doctor’s utterances are filtered:
 - a. Minimum: 10 words
 - b. Maximum: 100 words
 (to exclude uninformative short answers or overly long explanations).

```
def load_and_filter_dataset(path):
    with open(path, "r") as f:
        data = json.load(f)

    cleaned_data = []
    for entry in data:
        utterances = entry.get("utterances", [])
        if len(utterances) < 2:
            continue
        patient = utterances[0].replace("patient:", "").strip()
        doctor = utterances[1].replace("doctor:", "").strip()

        if 10 < len(doctor.split()) < 100: # filter bad/short/long responses
            input_text = f"You are a helpful telehealth doctor. Provide a short, clear response.\nPatient: {patient}\nDoctor:"
            cleaned_data.append({"input": input_text, "output": doctor})

    return pd.DataFrame(cleaned_data)
```

The provided code snippet demonstrates loading and cleaning the MedDialog dataset, specifically for a telehealth doctor’s context. It structures the data into 'input' (prompt for the model) and 'output' (desired doctor response) fields, adhering to the instruction-following format.

Data Splitting

It is essential to divide the dataset into three parts:

1. **Training Set:** Used to train the model.
2. **Validation Set:** Used to monitor the model’s performance during training and tune hyperparameters.
3. **Test Set:** Used for a final, unbiased evaluation of the model’s performance after training is complete.

```
# Load datasets
train_df = load_and_filter_dataset("english-train.json")
val_df = load_and_filter_dataset("english-dev.json")
test_df = load_and_filter_dataset("english-test.json")

# Split 20% of train data for validation (benchmarking)
train_df, benchmark_df = train_test_split(train_df, test_size=0.2, random_state=42)

dataset = DatasetDict({
    "train": Dataset.from_pandas(train_df),
    "benchmark": Dataset.from_pandas(benchmark_df), # Benchmark dataset (20%)
    "test": Dataset.from_pandas(test_df)
})
```

The provided code snippet demonstrates the dataset is split into training, validation and test sets. The 'benchmark' set is a 20% subset of the original training data, reserved for evaluating model performance during development and hyperparameter tuning without touching the final test set.

Therefore, FLAN-T5's strength lies in its instruction-following capabilities. Structuring our data to mimic instructions helps the model learn the desired behavior. Proper data splitting prevents the model from simply memorising the training data, leading to better generalisation. Using a dedicated 'benchmark' set is the best practice for robust evaluation.

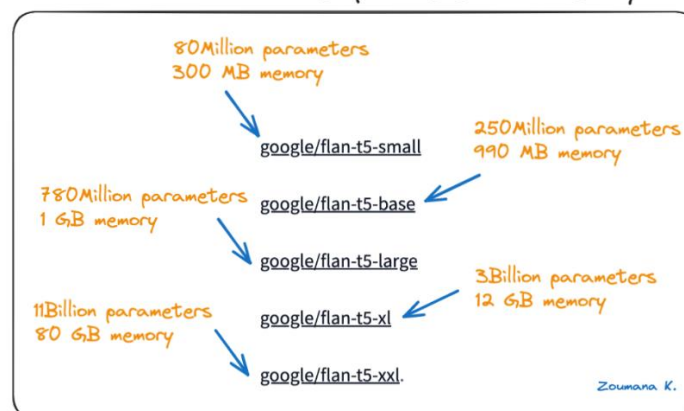
Step 3. Model Selection and Loading

Loading the FLAN-T5 Model

Multiple formats of FLAN-T5 models are available on Hugging Face, from small to extra-large models, and the bigger the model, the more parameters it has.

Below are the different model sizes available from the Hugging Face model card:

FLAN-T5 variants with their parameters and memory usage



Choosing the right model size

The choice of the right model size among the variants of FLAN-T5 highly depends on the following criteria:

- The specific requirements of the project
- The available computational resources
- The level of performance expected

The choice depends on our own computational resources (GPU memory, processing power) and the complexity of the task. Smaller models train faster and require less memory but might offer lower accuracy. Larger models generally provide better performance but demand more significant computational resources.

In this tutorial, a T4 GPU in Google Colab is being used for this experimentation, and the google/flan-t5-base model will strike a balance between computational efficiency and performance compatibility.

Model and Tokenizer initialization

The following is required to create the model.

```
from transformers import T5Tokenizer

model_name = "google/flan-t5-base" # Load the tokenizer
tokenizer = T5Tokenizer.from_pretrained(model_name) # Load the pretrained FLAN-T5 model
```

- The tokenizer is instantiated using the T5Tokenizer module and the name of the model

Selecting the appropriate model size is a critical early decision. It impacts training time, memory usage, and ultimately, the performance ceiling of the fine-tuned model. Starting with a smaller model is often a good strategy for initial experimentation and debugging.

Data formatting and tokenisation - Preprocessing

Each text input and target need to be tokenised. This converts the text into numerical representations that the model can understand. We will use the specific tokeniser associated with our chosen FLAN-T5 model. It is important to ensure consistent tokenisation, and to handle padding (adding special tokens to make sequences the same length) and truncation (cutting off sequences that are too long) appropriately.

```
def preprocess(example):
    inputs = tokenizer(example["input"], padding="max_length", truncation=True, max_length=512)
    targets = tokenizer(example["output"], padding="max_length", truncation=True, max_length=128)
    inputs["labels"] = [t if t != tokenizer.pad_token_id else -100 for t in targets["input_ids"]]
    return inputs

tokenized_dataset = dataset.map(preprocess, batched=True)
```

So, the code snippet demonstrates that we use the T5 tokenizer from Hugging Face.

- Input sequence length: 512 tokens
- Output sequence length: 128 tokens
- Special handling: padding tokens in labels are replaced with -100 so they are ignored in loss computation.

Correct preprocessing ensures that the model receives data in a format it can process efficiently.

Step 4: Fine-Tuning Strategy

The Challenge of Full Fine-tuning and the Rise of PEFT

While fine-tuning is powerful, directly fine-tuning all the parameters of very large models like FLAN-T5 can be computationally prohibitive. It demands significant GPU memory, processing power, and storage for the updated model weights. This is where Parameter-Efficient Fine-Tuning (PEFT) techniques come into play. PEFT methods aim to achieve comparable performance to full fine-tuning while updating only a small fraction of the model's parameters, or by adding a small number of new trainable parameters, thus drastically reducing computational cost and memory footprint.

Parameter-Efficient Fine-Tuning (PEFT) and LoRA

Among the various PEFT techniques, LoRA (Low-Rank Adaptation) is a particularly effective and popular choice. LoRA is used because it offers a smart way to adapt large models efficiently. It works by injecting small, trainable low-rank matrices into specific layers of the pre-trained model, typically the attention mechanisms. During the fine-tuning process, only these newly introduced low-rank matrices are updated, while most of the original FLAN-T5 model's weights remain frozen. This approach is highly efficient as it significantly reduces the number of parameters that need to be trained.

LoRA Configuration

Key Arguments Explained:

1. **r (Rank):** This parameter defines the rank of the low-rank matrices that LoRA adds. A higher rank means more trainable parameters and potentially more expressiveness, but also higher computational cost. Common values range from 4 to 64.
2. **lora_alpha (Scaling Factor):** This is a scaling factor for the LoRA weights. The update is scaled by $\text{lora_alpha} / r$. It helps to control the magnitude of adaptation. Setting it to twice the rank ($\text{lora_alpha}=32$ for $r=16$) is a common practice.
3. **lora_dropout (Dropout Rate):** This applies dropout to the LoRA layers to prevent overfitting. A value like 0.1 (10%) is a standard regularization technique.
4. **target_modules (Modules to Adapt):** This specifies which layers or modules within the pre-trained model should have LoRA matrices injected. For transformer models like T5, common targets include the query (q), key (k), value (v), output (o) projections, and sometimes the feed-forward network layers (w_i , w_o). Targeting these specific modules is based on research indicating they are critical for adapting the model's behavior. The configuration targets ["q", "v", "k", "o", "wi", "wo"], which is a comprehensive selection for T5 architectures.

5. **bias (Bias Handling):** This parameter determines how the bias terms in the model are handled. Setting it to “none” means that LoRA will not be applied to bias parameters, simplifying the adaptation process and often yielding good results. Other options might include learning the bias or not adapting it.
6. **task_type (Task Type):** This tells PEFT what kind of task the model is being fine-tuned for. `TaskType.SEQ_2_SEQ_LM` is appropriate for models like T5 that are designed for sequence-to-sequence tasks (translation, summarization, text generation where input and output are sequences).

```
from transformers import T5ForConditionalGeneration
from peft import get_peft_model, LoraConfig, TaskType

model = T5ForConditionalGeneration.from_pretrained(model_name)

lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.1,
    target_modules=["q", "v", "k", "o", "wi", "wo"],
    bias="none",
    task_type=TaskType.SEQ_2_SEQ_LM
)

model = get_peft_model(model, lora_config)
```

The purpose of using LoRA for fine-tuning FLAN-T5 is to efficiently adapt this powerful pre-trained model to the specific downstream task without the prohibitive costs associated with full fine-tuning. The justification for this approach lies in only training a small number of parameters (those within the low-rank matrices), LoRA drastically reduces GPU memory requirements and training time. This makes it feasible to fine-tune large models on consumer-grade hardware. By keeping the original model’s weights frozen, LoRA helps to preserve the general knowledge learned during pre-training, reducing the risk of the model forgetting its core capabilities while learning the new task.

Therefore, the PEFT LoRA configuration set up is an efficient way to leverage the power of FLAN-T5 for specific needs, making advanced AI capabilities more accessible and manageable.

Step 5. Training Configuration

The provided `TrainingArguments` from the **transformers** library outline key parameters for fine-tuning FLAN-T5.

Key Arguments Explained:

1. **output_dir="/flan-t5-base-telehealth-lora":** This specifies the directory where the model checkpoints, logs, and other outputs will be saved during and after training. It is crucial for managing experiments and retrieving the trained model.

2. **eval_strategy="epoch"**: This setting dictates that model evaluation will be performed at the end of each training epoch. Evaluating periodically helps monitor the model's progress, detect overfitting, and decide when to stop training.
3. **save_strategy="epoch"**: Like `eval_strategy`, this ensures that the model's state (checkpoints) is saved at the end of each epoch. This is vital for resuming training if interrupted and for keeping track of the best-performing model versions.
4. **learning_rate=2e-4**: The learning rate controls the step size during model optimization. A value of 2e-4 is for fine-tuning Transformer models, balancing the speed of convergence with the risk of overshooting optimal weights. It is a critical hyperparameter that significantly impacts training stability and performance.
5. **per_device_train_batch_size=4**: This defines the number of samples processed by each GPU (or CPU) in a single training step. A batch size of 4 is relatively small, which can help with memory constraints on individual devices and sometimes lead to more stable gradient updates, especially with limited hardware.
6. **per_device_eval_batch_size=4**: Similar to the training batch size, this sets the batch size for evaluation. Smaller evaluation batch sizes can also help manage memory and provide more granular insights into performance during the evaluation phase.
7. **num_train_epochs=30**: An epoch represents one full pass through the entire training dataset. Setting this to 30 epochs suggests a thorough training process. The optimal number of epochs depends on the dataset size and complexity, and early stopping based on evaluation performance is often used to prevent overfitting.
8. **weight_decay=0.01**: Weight decay is a regularization technique that penalizes large weights in the model, helping to prevent overfitting. A value of 0.01 is a moderate regularization strength.
9. **logging_steps=20**: This determines how often training metrics (like loss and learning rate) are logged. Logging every 20 steps provides frequent updates on the training progress without overwhelming the log output.
10. **report_to="none"**: This indicates that no external experiment tracking platforms (like W&B or TensorBoard) are being used. All reporting will be local via logs.
11. **save_total_limit=2**: This parameter limits the total number of checkpoints saved. By setting it to 2, only the two most recent checkpoints are kept, saving disk space. This

is useful when disk space is limited or when we only need to keep track of recent model states.

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./flan-t5-base-telehealth-lora",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-4,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=30,
    weight_decay=0.01,
    logging_steps=20,
    report_to="none",
    save_total_limit=2
)
```

Carefully configuring these hyperparameters is crucial. They dictate the learning process, affecting convergence speed, model stability, and the final performance. Experimenting these values is often necessary to find the optimal settings for the specific task and dataset.

Step 6. Model Training

The Hugging Face **Trainer** class simplifies the entire training and evaluation process. It handles the training loop, optimizer, learning rate scheduler, gradient accumulation, mixed-precision training, and evaluation.

Trainer Initialization: The **Trainer** is initialized with the model, the defined training arguments, the tokenized training and validation datasets, the tokenizer, and the data collator.

Training Execution: Simply call the `trainer.train()` method to start the fine-tuning process. The trainer will manage all aspects of training, including saving checkpoints based on the `TrainingArguments`.

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["benchmark"], # Evaluation on benchmark dataset
    tokenizer=tokenizer
)

trainer.train()
model.save_pretrained("./flan-t5-base-telehealth-lora")
```

The **Trainer** class is a powerful abstraction that significantly reduces the amount of code needed to manage a deep learning training process. It ensures best practices are followed (like handling device placement and mixed precision) and streamlines the workflow,

allowing them to focus more on the data and model. Saving the PEFT model (which only contains the adapter weights) is much more efficient than saving the entire base model.

Training Epoch Result

Let’s understand what the training epoch of the model performance.

Epoch	Training Loss	Validation Loss
1	1.232500	0.922221
2	0.888600	0.833157
3	0.786000	0.789624
4	0.898400	0.761712
5	0.756300	0.741800
6	0.710400	0.728316
7	0.907600	0.710487
8	0.693800	0.699710
9	0.795400	0.689593
10	0.641500	0.681998
11	0.723800	0.675486
12	0.596600	0.671910
13	0.621700	0.666521
14	0.585400	0.664083
15	0.540200	0.663053
16	0.739200	0.658193
17	0.554600	0.653566
18	0.721600	0.650172
19	0.557800	0.646249
20	0.732800	0.646254
21	0.542700	0.643977
22	0.600500	0.644585
23	0.615600	0.642109
24	0.526600	0.641755
25	0.675200	0.642162
26	0.645500	0.641449
27	0.456700	0.639812
28	0.797100	0.640480
29	0.551700	0.639781
30	0.537400	0.640555

Training Loss and Validation Loss: Lower values in these metrics are preferable as they indicate a better fit of the model to the data.

Result Observed

The FLAN-T5-Base model was fine-tuned over 30 epochs using a per-device batch size of 4 and a learning rate of 2e-4, with weight decay set to 0.01. Evaluation and checkpoint saving were performed at the end of each epoch, and the training process was logged every 20 steps. The model was trained using the Hugging Face Trainer API on a domain-specific telehealth dataset, with evaluation conducted on a benchmark subset. Both training and validation loss steadily declined throughout training, reaching approximately 0.54 and 0.64 respectively by the final epoch. The consistent downward trend in both metrics, coupled with the absence of divergence between them, indicates that the model did not overfit and maintained strong generalisation to unseen patient queries. The training configuration featuring a moderately high learning rate, weight decay, and evaluation-based checkpointing enabled stable convergence on a T4 GPU. Therefore, the fine-tuning process yielded a robust and efficient model suitable for real-world deployment in telehealth.

Step 7. Evaluation Metrics

Importance of Evaluation Metrics

Evaluating the performance of a fine-tuned language model is crucial to understand how well it has learned the desired task and to compare its effectiveness against a baseline or other models. In the context of text generation, such as in a telehealth scenario, metrics help quantify the quality of the generated responses.

Evaluation Metrics Used: ROUGE and BERTScore

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

What it is: ROUGE is a set of metrics used for evaluating automatic summarization and machine translation tasks. It measures the overlap of n-grams (sequences of words) between the machine-generated text and human-written reference texts.

Techniques Used: The most common variants are:

- **ROUGE-N:** Measures the overlap of n-grams. ROUGE-1 uses unigrams (single words), ROUGE-2 uses bigrams (pairs of words), and ROUGE-L uses the longest common subsequence.
- **ROUGE-L:** Captures sentence-level structure similarity by considering the longest common subsequence of words between the prediction and the reference.

Why it is used: ROUGE is effective at capturing the content overlap and fluency of generated text. For summarization and response generation, it helps determine if the generated text contains the key information present in the reference.

Purpose: To assess the quality of the generated responses by measuring their similarity to the ground truth (human-written) responses in terms of content and phrasing. Higher ROUGE scores generally indicate better quality.

BERTScore

What it is: BERTScore is an advanced evaluation metric that leverages contextual embeddings from BERT to compute the similarity between tokens in the generated text and the reference text. Unlike ROUGE, which relies on exact word matches, BERTScore captures semantic similarity.

Techniques Used: It computes the cosine similarity between token embeddings of the prediction and the reference. It then matches tokens greedily and computes precision, recall, and F1-score based on these similarities.

Why it is used: BERTScore is preferred when semantic understanding is important. It can identify paraphrased or semantically similar phrases that ROUGE might miss because they don't share exact word overlap. This is particularly useful in conversational AI or tasks where understanding the meaning is paramount.

Purpose: To provide a more nuanced evaluation of the generated text by considering the semantic meaning rather than just surface-level word overlap. It helps ensure that the generated responses are not only factually similar but also semantically coherent and relevant to the context.

Key Libraries and Setup

Begin with importing necessary libraries. **torch** is for tensor operations and device management (CPU/GPU). The **evaluate** library from Hugging Face provides easy access to various NLP metrics, and **datasets** are used for efficient data handling. We specify the device to leverage a GPU if available for faster processing. The **rouge** and **bertscore** metrics are loaded from the **evaluate** library.

```
import torch
from evaluate import load
from datasets import Dataset

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
rouge = load("rouge")
bertscore = load("bertscore")
```

Using specialized libraries like **evaluate** simplifies the implementation of complex metrics. Explicitly setting the device ensures that the computation runs on the most efficient hardware available, which is crucial for LLM evaluation.

The `evaluate_model` function

This function encapsulates the core logic for generating predictions and computing evaluation metrics. It takes the model and a dataset as input. It iterates through each example, formats a specific prompt tailored for a telehealth doctor, generates a response using the provided model, and then decodes the output. Finally, it computes ROUGE and BERTScore using the generated predictions and the ground truth references.

Key Components used:

Prompt Formatting

A specific prompt structure (You are a helpful telehealth doctor. Provide a short, clear response. Patient: {patient_text} Doctor:) is used. This is essential for guiding the model to behave as a telehealth doctor and produce relevant, concise responses.

Tokenisation and Generation

The input prompt is tokenized. The **model.generate()** method is then called with several parameters to control the output quality:

1. **max_new_tokens=128**: Limits the generated response to a maximum of 128 new tokens.
2. **num_beams=4**: Uses beam search with 4 beams, exploring multiple generation paths to find a higher-quality output.
3. **early_stopping=True**: Stops generation when all beam hypotheses have reached the end-of-sequence token.
4. **repetition_penalty=1.2**: Discourages the model from repeating itself by penalizing tokens that have already appeared.
5. **no_repeat_ngram_size=3**: Prevents the model from repeating trigrams (sequences of three words).

Decoding

The model's output token IDs are converted back into human-readable text using **tokenizer.decode**.

Metric Computation

- **ROUGE**: calculates ROUGE scores (ROUGE-1, ROUGE-2, ROUGE-L) by comparing the generated ``preds`` with the ground truth ``refs``.
- **BERTScore**: computes BERTScore, which leverages contextual embeddings to measure semantic similarity. ``lang="en"`` specifies the language for embedding calculation.
- **BERTScore Averaging**: averages the F1 scores (``sum(bert_result["f1"]) / len(bert_result["f1"])``) to provide a single, representative BERTScore metric for the entire set of predictions.

```
def evaluate_model(model, eval_df):
    preds, refs = [], []

    # Convert the Dataset to a pandas DataFrame
    eval_df = eval_df.to_pandas()

    for i, row in eval_df.iterrows():
        patient_text = row["input"].split("Patient:")[1].split("\n")[0].strip()
        prompt = f"You are a helpful telehealth doctor. Provide a short, clear response.\nPatient: {patient_text}\nDoctor:"
        inputs = tokenizer(prompt, return_tensors="pt", truncation=True, padding="max_length", max_length=512).to(device)
        outputs = model.generate(
            **inputs,
            max_new_tokens=128,
            num_beams=4,
            early_stopping=True,
            repetition_penalty=1.2,
            no_repeat_ngram_size=3
        )
        preds.append(tokenizer.decode(outputs[0], skip_special_tokens=True))
        refs.append(row["output"])

    # Evaluate ROUGE
    rouge_result = rouge.compute(predictions=preds, references=refs)

    # Evaluate BERTScore
    bert_result = bertscore.compute(predictions=preds, references=refs, lang="en")

    return rouge_result, sum(bert_result["f1"]) / len(bert_result["f1"])
```

This function encapsulates the evaluation logic, making it reusable for both the pre-trained and fine-tuned models. The specific prompt and generation parameters are crucial for eliciting high-quality, relevant responses tailored to the telehealth task. ROUGE and BERTScore are chosen to provide a comprehensive evaluation: ROUGE measures lexical overlap (important for conciseness and factual recall), while BERTScore assesses semantic similarity, capturing nuances in meaning that lexical overlap might miss.

Pipeline Integration and Comparison

In the provided pipeline, both ROUGE and BERTScore are computed for both the pretrained FLAN-T5 model and the LoRA fine-tuned model. This allows for a direct comparison:

Pretrained Model: Evaluates the base performance of FLAN-T5 before any task-specific adaptation.

LoRA Fine-Tuned Model: Assesses the improvement gained after fine-tuning with LoRA on the telehealth dataset.

By comparing the scores (ROUGE-1, ROUGE-2, ROUGE-L, and BERTScore-F1), one can quantitatively determine if the fine-tuning process has led to better response generation, indicating whether the model has learned to produce more relevant, informative, and semantically accurate outputs for the telehealth domain.


```

# Evaluate LoRA fine-tuned model
print("Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...")
lora_model = T5ForConditionalGeneration.from_pretrained("./flan-t5-base-telehealth-lora").to(device)
lora_rouge, lora_bert = evaluate_model(lora_model, dataset["benchmark"])

# Performance comparison (Before vs After Fine-Tuning)
import pandas as pd

comparison_df = pd.DataFrame([
    "Metric": ["ROUGE-1", "ROUGE-2", "ROUGE-L", "BERTScore-F1"],
    "Pretrained": [
        pre_rouge["rouge1"],
        pre_rouge["rouge2"],
        pre_rouge["rougeL"],
        pre_bert,
    ],
    "LoRA Fine-Tuned": [
        lora_rouge["rouge1"],
        lora_rouge["rouge2"],
        lora_rouge["rougeL"],
        lora_bert,
    ]
])

```

These metrics are essential for objectively assessing how closely the model's generated responses align with the desired or reference outputs. They quantify the success of the fine-tuning process by providing measurable improvements in relevance, coherence, and semantic accuracy.

Evaluation Result

The LoRA fine-tuned FLAN-T5-Base model demonstrated substantial improvements across all key metrics when evaluated on the benchmark dataset, compared to the original pretrained version. These metrics, ROUGE-1, ROUGE-2, ROUGE-L, and BERTScore-F1, collectively assess the model's ability to generate accurate, fluent, and semantically meaningful responses.

```

Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...
Metric  Pretrained  LoRA Fine-Tuned
0      ROUGE-1    0.047819    0.414246
1      ROUGE-2    0.006027    0.218925
2      ROUGE-L    0.041879    0.366255
3  BERTScore-F1    0.837975    0.902521

```

ROUGE-1 / ROUGE-2 / ROUGE-L

These metrics measure n-gram overlap between generated responses and reference answers. The dramatic increase in ROUGE scores, especially ROUGE-2 (from 0.006 to 0.219) indicates that the fine-tuned model produces outputs that are far more aligned with expected phrasing and structure. This is critical in telehealth contexts, where precision and clarity are essential.

BERTScore-F1

This semantic similarity metric evaluates how close the generated text is to the reference in meaning, using contextual embeddings. The jump from 0.838 to 0.903 reflects a significant gain in the model's ability to understand and reproduce medically relevant information with greater semantic fidelity.

The fine-tuned model's performance suggests that LoRA adaptation was highly effective in transferring domain-specific knowledge without requiring full retraining. The improvements across both lexical (ROUGE) and semantic (BERTScore) metrics confirm that the model not only learned to mimic surface-level patterns but also internalized deeper contextual understanding of patient queries and responses.

This is particularly valuable in telehealth, where both linguistic accuracy and semantic relevance are vital. The pretrained model, while capable, lacked domain alignment evident in its low ROUGE scores. Fine-tuning with LoRA bridged this gap efficiently, yielding a model that is better suited for real-world deployment in telehealth scenarios.

Step 8. Deployment (Interactive Chat with the Model)

Model inference is the critical final stage where the fine-tuned model is used to generate predictions or outputs based on new, unseen input data. For the FLAN-T5 fine-tuning pipeline, the importance, purpose, and justification of model inference are:

To leverage the specialized knowledge and capabilities acquired during fine-tuning. In the context of the telehealth tutorial, the purpose is to enable the FLAN-T5 model to act as a helpful and informative Telehealth Assistant, responding to patient queries accurately and appropriately.

Inference is where the model's value is realised. Without inference, the fine-tuning process would be pointless, as the model wouldn't be applied to solve real-world problems. It's the stage that demonstrates the model's effectiveness and utility.

The comprehensive fine-tuning pipeline is justified by its ability to produce a model capable of performing specific tasks with high accuracy and efficiency. For instance, a fine-tuned FLAN-T5 for telehealth can assist healthcare professionals, improve patient engagement, and provide quick access to information, thereby enhancing the overall healthcare experience. The interactive chat example showcases this by allowing users to simulate conversations with the AI assistant, demonstrating its practical application in a controlled environment before real-world deployment.

```

# Step 5: Interactive Chat with the model
def chat_with_model(model, tokenizer):
    print("\nTelehealth Assistant is ready.")
    print("Please choose a prompt mode:")
    print("1 - General")
    print("2 - Diagnosis")
    print("3 - Psychological")
    print("4 - Medication")

    mode_map = {
        "1": "general",
        "2": "diagnosis",
        "3": "psychological",
        "4": "medication"
    }

    mode = input("Enter mode number (1-4): ").strip()
    prompt_type = mode_map.get(mode, "general") # default to general

    print(f"\nPrompt mode set to [{prompt_type.upper()}]. Type 'exit' to quit.\n")

    while True:
        user_input = input("Patient: ").strip()
        if user_input.lower() == "exit":
            print("Thank you. Stay healthy!")
            break
        doctor_reply = generate_answer(user_input, model, tokenizer, prompt_type=prompt_type)
        print("Doctor:", doctor_reply, "\n")

# Start the chat system
chat_with_model(model, tokenizer)

```

A user-facing popup that lets users choose from four preset modes: General, Diagnosis, Psychological, and Medication opens the deployment. A `mode_map` dictionary is used to manage this, mapping numeric input to the appropriate prompt type. By directing the model's generating behaviour towards the desired clinical domain, this architecture makes sure that the model receives context-specific instructions.

The system begins a conversational loop when a prompt mode is chosen. The `generate_answer()` method receives patient-style enquiries from the user and the prompt type they have chosen. After that, the model responds in the manner of a doctor. The user must write "exit" to end the loop, which guarantees a smooth and regulated interaction flow. Safe termination, extension, and repeatability are all supported by this modular framework.

Prompt mapping, input processing, and inference logic are separated to facilitate simple expansion. Subsequent versions could have multi-turn memory, logging systems, or GUI components. The existing CLI format is portable and perfect for clinical NLP workflow testing, demonstrations, and prototyping.

Telehealth Assistant is ready.
Please choose a prompt mode:
1 - General
2 - Diagnosis
3 - Psychological
4 - Medication
Enter mode number (1-4): 1

Prompt mode set to [GENERAL]. Type 'exit' to quit.

Patient: Hey Doc, I've been sneezing nonstop, do you think it's just allergies?
Doctor: it may be allergies. Stay hydrated, use antihistamines if needed, and monitor symptoms. Seek care if symptoms persist or worsen.

Patient: I've had a cough and low-grade fever for three days. Should I be worried?
Doctor: this may be a mild viral infection. Rest, hydrate, and monitor. Seek care if symptoms worsen or persist.

Patient: I just twisted my ankle while jogging. It's swollen but not bruised, should I get an X-ray?
Doctor: swollen ankle. Rest, ice, compression, and compression may help. Seek medical attention if swelling worsens, blisters, or is severe.

Patient: Is it okay to exercise if I have a cold?
Doctor: if you are not able to exercise, rest, hydrate, and monitor symptoms. If symptoms persist, we will consider antihistamines or physical therapy.

Patient: I have a sore throat and mild fever. What should I do?
Doctor: rest, hydrate, and monitor. Seek care if symptoms worsen.

Patient: exit
Thank you. Stay healthy!

The model exhibits good generalisation and clinical appropriateness when used in the General prompt mode. The responses are careful, context-aware, and medically appropriate; they place a strong emphasis on hydration, symptom monitoring, and escalation protocols. For instance:

1. Symptoms of allergies: The model suggests taking antihistamines and staying hydrated, but it also advises getting help if symptoms don't go away.
2. Viral infection: It advises monitoring and relaxation rather than drawing hasty judgements.
3. Ankle injury: The model signals serious symptoms for medical treatment and uses the RICE technique, albeit with some repetition ("compression and compression").
4. Exercise when sick: It recommends relaxation and symptom monitoring while striking a balance between prudence and adaptability.
5. Fever and sore throat: The same advice applies take it easy, drink plenty of water, and get help if you need it.

These answers demonstrate conformity to telehealth standards and support the model's capacity to produce outputs that are secure and easy to understand.

The method places more emphasis on symptom management and backup plans than it does on overconfident diagnosis. In clinical deployment, where safety and clarity are crucial, this behaviour is essential. While prompt mode selection allows for focused review across domains, the exit command guarantees user autonomy.

The model's preparedness for interaction in the actual world is confirmed by this deployment stage. It demonstrates that the model can react suitably to a range of patient enquiries and that quick engineering successfully shapes inference behaviour. The technology may be used to prototype chatbots tailored to a certain subject, instructional resources, or healthcare assistants. Improvements like input validation, response filtering, and clinical supervision would be required for production usage to guarantee compliance and safety.

Hyperparameter Tuning Test Cases and Analysis

Hyperparameter tuning is essential to the success of adapting large language models with parameter-efficient fine-tuning strategies such as LoRA (Low-Rank Adaptation). Because LoRA limits training to a narrow number of rank-decomposition matrices, the selection of hyperparameters has a disproportionately large influence compared to complete fine-tuning, which updates billions of parameters.

I systematically changed the optimization parameters (learning rate, batch size, weight decay), training length (epochs), and LoRA setup (r, alpha, dropout, and targeted modules) over 17 test cases. Evaluation metrics assessed using BERTScore-F1 for semantic fidelity and ROUGE (1, 2, L) for surface-level correctness. The discussion that follows evaluates these situations critically, contrasts the results, and summarizes the ideas that guided the configuration that was ultimately selected.

Test Case 1

This configuration establishes a minimal baseline by setting a very low LoRA rank (r=4) with a modest scaling factor (alpha=8) and adapting only the query (q) and value (v) projection matrices. The intention here was to test whether such a lightweight and parameter-efficient setup could still bring measurable improvements in task adaptation. By excluding key (k) and output (o) modules, the model’s representational adjustments are intentionally constrained. This baseline helps clarify the lower bound of LoRA effectiveness, showing how much capacity is needed before meaningful gains can be observed.

PEFT LoRA Configuration	Training Configuration															
<pre>lora_config = LoraConfig(r=4, lora_alpha=8, lora_dropout=0.0, target_modules=["q", "v"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-5, per_device_train_batch_size=2, per_device_eval_batch_size=2, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>															
Model Evaluation Result																
<pre>Evaluating LoRA Fine-Tuned Model...</pre>																
	<table><tr><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0 ROUGE-1</td><td>0.082759</td><td>0.080950</td></tr><tr><td>1 ROUGE-2</td><td>0.010410</td><td>0.008178</td></tr><tr><td>2 ROUGE-L</td><td>0.065898</td><td>0.063763</td></tr><tr><td>3 BERTScore-F1</td><td>0.831523</td><td>0.829021</td></tr></table>	Metric	Pretrained	LoRA Fine-Tuned	0 ROUGE-1	0.082759	0.080950	1 ROUGE-2	0.010410	0.008178	2 ROUGE-L	0.065898	0.063763	3 BERTScore-F1	0.831523	0.829021
Metric	Pretrained	LoRA Fine-Tuned														
0 ROUGE-1	0.082759	0.080950														
1 ROUGE-2	0.010410	0.008178														
2 ROUGE-L	0.065898	0.063763														
3 BERTScore-F1	0.831523	0.829021														

Test Case 2

In this setup, LoRA rank is increased to $r=8$, and coverage is extended across all attention projections (q, k, v, o). The aim was to evaluate whether breadth of adaptation compensates for the relatively small rank. This design probes whether attending to more subcomponents of the attention mechanism leads to richer contextual adaptation.

PEFT LoRA Configuration	Training Configuration															
<pre>lora_config = LoraConfig(r=8, lora_alpha=8, lora_dropout=0.0, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-5, per_device_train_batch_size=2, per_device_eval_batch_size=2, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>															
Model Evaluation Result																
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																
	<table><tr><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0 ROUGE-1</td><td>0.094097</td><td>0.082897</td></tr><tr><td>1 ROUGE-2</td><td>0.012672</td><td>0.009427</td></tr><tr><td>2 ROUGE-L</td><td>0.072715</td><td>0.065417</td></tr><tr><td>3 BERTScore-F1</td><td>0.828830</td><td>0.822476</td></tr></table>	Metric	Pretrained	LoRA Fine-Tuned	0 ROUGE-1	0.094097	0.082897	1 ROUGE-2	0.012672	0.009427	2 ROUGE-L	0.072715	0.065417	3 BERTScore-F1	0.828830	0.822476
Metric	Pretrained	LoRA Fine-Tuned														
0 ROUGE-1	0.094097	0.082897														
1 ROUGE-2	0.012672	0.009427														
2 ROUGE-L	0.072715	0.065417														
3 BERTScore-F1	0.828830	0.822476														

Test Case 3

Here, the rank remains at $r=8$, but the scaling factor (alpha) has doubled from 8 to 16. The purpose was to investigate whether amplifying the strength of the LoRA updates allows more effective gradient flow and signal propagation, even when rank is modest. Unlike Test Case 2, this setup focuses on depth of influence rather than breadth, keeping the same modules while increasing the impact per update. It effectively tests the hypothesis that higher alpha can partially offset low rank, improving learning without adding significant parameter cost.

PEFT LoRA Configuration	Training Configuration
<pre>lora_config = LoraConfig(r=8, lora_alpha=16, lora_dropout=0.0, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-5, per_device_train_batch_size=2, per_device_eval_batch_size=2, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>

Model Evaluation Result				
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...				
	Metric	Pretrained	LoRA	Fine-Tuned
0	ROUGE-1	0.094097		0.090996
1	ROUGE-2	0.012672		0.013367
2	ROUGE-L	0.072715		0.072687
3	BERTScore-F1	0.828830		0.822629

Test Case 4

This case scales up both rank (32) and alpha (32) while introducing dropout at 0.1. The design reflects the hypothesis that larger adapter capacity would better capture domain-specific task nuances, while dropout would help avoid overfitting. By targeting all attention projections, this setup marks the first high-capacity configuration in the series. The intent was to establish whether “bigger plus regularized” outperforms the leaner setups tested earlier.

PEFT LoRA Configuration	Training Configuration																				
<pre>lora_config = LoraConfig(r=32, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-5, per_device_train_batch_size=2, per_device_eval_batch_size=2, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>																				
Model Evaluation Result																					
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																					
	<table><tr><th></th><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0</td><td>ROUGE-1</td><td>0.094097</td><td>0.100013</td></tr><tr><td>1</td><td>ROUGE-2</td><td>0.012672</td><td>0.012184</td></tr><tr><td>2</td><td>ROUGE-L</td><td>0.072715</td><td>0.076774</td></tr><tr><td>3</td><td>BERTScore-F1</td><td>0.828830</td><td>0.820528</td></tr></table>		Metric	Pretrained	LoRA Fine-Tuned	0	ROUGE-1	0.094097	0.100013	1	ROUGE-2	0.012672	0.012184	2	ROUGE-L	0.072715	0.076774	3	BERTScore-F1	0.828830	0.820528
	Metric	Pretrained	LoRA Fine-Tuned																		
0	ROUGE-1	0.094097	0.100013																		
1	ROUGE-2	0.012672	0.012184																		
2	ROUGE-L	0.072715	0.076774																		
3	BERTScore-F1	0.828830	0.820528																		

Test Case 5

Building on Case 4, dropout is increased to 0.2 while keeping rank and alpha high. This tests whether moderate regularization further stabilizes performance in a high-capacity adapter. The design assumes that slightly stronger dropout could strike a balance between learning power and generalisation, particularly in preventing over-reliance on a few dominant features. By comparing Test Case 4, this case directly measures the effect of regularization strength on LoRA fine-tuning.

PEFT LoRA Configuration	Training Configuration		
<pre>lora_config = LoraConfig(r=32, lora_alpha=32, lora_dropout=0.2, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-5, per_device_train_batch_size=2, per_device_eval_batch_size=2, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>		
Model Evaluation Result			
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...			
	Metric	Pretrained	LoRA Fine-Tuned
0	ROUGE-1	0.094097	0.098676
1	ROUGE-2	0.012672	0.013119
2	ROUGE-L	0.072715	0.078817
3	BERTScore-F1	0.828830	0.823699

Test Case 6

Dropout increased further to 0.3, representing aggressive regularization. The motivation was to test whether higher dropout prevents overfitting in LoRA’s expanded configuration, or whether it suppresses the model’s capacity to adapt effectively. Since high dropout reduces the effective learning signal per iteration, this case explores whether too much regularization becomes counterproductive. It effectively probes the upper bound of dropout tolerance for high-rank LoRA adapters.

PEFT LoRA Configuration	Training Configuration
<pre>lora_config = LoraConfig(r=32, lora_alpha=32, lora_dropout=0.3, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-5, per_device_train_batch_size=2, per_device_eval_batch_size=2, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>

Model Evaluation Result				
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...				
	Metric	Pretrained	LoRA	Fine-Tuned
0	ROUGE-1	0.094097	0.095695	
1	ROUGE-2	0.012672	0.013782	
2	ROUGE-L	0.072715	0.076412	
3	BERTScore-F1	0.828830	0.823748	

Test Case 7

This setup reduces rank to r=16 while keeping alpha high (32) and dropout moderate (0.1). The rationale was to explore a balanced mid-capacity configuration, testing whether lower parameter cost with tuned regularization can deliver comparable performance to full high-rank setups. This experiment reflects a practical trade-off scenario, where resource efficiency is considered alongside accuracy. It also provides a checkpoint between the minimal and maximal LoRA ranks tested earlier.

PEFT LoRA Configuration	Training Configuration																				
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-5, per_device_train_batch_size=2, per_device_eval_batch_size=2, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>																				
Model Evaluation Result																					
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																					
	<table><tr><th></th><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0</td><td>ROUGE-1</td><td>0.094097</td><td>0.104711</td></tr><tr><td>1</td><td>ROUGE-2</td><td>0.012672</td><td>0.012669</td></tr><tr><td>2</td><td>ROUGE-L</td><td>0.072715</td><td>0.082867</td></tr><tr><td>3</td><td>BERTScore-F1</td><td>0.828830</td><td>0.823547</td></tr></table>		Metric	Pretrained	LoRA Fine-Tuned	0	ROUGE-1	0.094097	0.104711	1	ROUGE-2	0.012672	0.012669	2	ROUGE-L	0.072715	0.082867	3	BERTScore-F1	0.828830	0.823547
	Metric	Pretrained	LoRA Fine-Tuned																		
0	ROUGE-1	0.094097	0.104711																		
1	ROUGE-2	0.012672	0.012669																		
2	ROUGE-L	0.072715	0.082867																		
3	BERTScore-F1	0.828830	0.823547																		

Test Case 8

For this case, learning rate is increased to 1e-4 and batch size doubled from 2 to 4, while keeping LoRA parameters fixed. The purpose was to examine whether stronger optimization dynamics accelerate convergence and yield better task adaptation. Increasing batch size provides more stable gradient estimates, while the higher learning rate allows faster updates. This design tests the interaction between optimization aggressiveness and model stability in LoRA fine-tuning.

PEFT LoRA Configuration	Training Configuration																				
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=1e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>																				
Model Evaluation Result																					
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																					
	<table><tr><th></th><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0</td><td>ROUGE-1</td><td>0.094097</td><td>0.198754</td></tr><tr><td>1</td><td>ROUGE-2</td><td>0.012672</td><td>0.036173</td></tr><tr><td>2</td><td>ROUGE-L</td><td>0.072715</td><td>0.137097</td></tr><tr><td>3</td><td>BERTScore-F1</td><td>0.828830</td><td>0.842268</td></tr></table>		Metric	Pretrained	LoRA Fine-Tuned	0	ROUGE-1	0.094097	0.198754	1	ROUGE-2	0.012672	0.036173	2	ROUGE-L	0.072715	0.137097	3	BERTScore-F1	0.828830	0.842268
	Metric	Pretrained	LoRA Fine-Tuned																		
0	ROUGE-1	0.094097	0.198754																		
1	ROUGE-2	0.012672	0.036173																		
2	ROUGE-L	0.072715	0.137097																		
3	BERTScore-F1	0.828830	0.842268																		

Test Case 9

Here, learning rate is further raised to 3e-4 while keeping batch size at 4. The experiment tests the hypothesis that more aggressive optimization could unlock additional performance gains by encouraging the model to escape shallow minima. The risk, however, is that too high a learning rate may cause unstable updates. This case is therefore positioned as a stress test of learning rate limits, building on Case 8's moderate increase.

PEFT LoRA Configuration	Training Configuration															
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=3e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>															
Model Evaluation Result																
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																
	<table><tr><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0 ROUGE-1</td><td>0.094097</td><td>0.205511</td></tr><tr><td>1 ROUGE-2</td><td>0.012672</td><td>0.040306</td></tr><tr><td>2 ROUGE-L</td><td>0.072715</td><td>0.143280</td></tr><tr><td>3 BERTScore-F1</td><td>0.828830</td><td>0.845428</td></tr></table>	Metric	Pretrained	LoRA Fine-Tuned	0 ROUGE-1	0.094097	0.205511	1 ROUGE-2	0.012672	0.040306	2 ROUGE-L	0.072715	0.143280	3 BERTScore-F1	0.828830	0.845428
Metric	Pretrained	LoRA Fine-Tuned														
0 ROUGE-1	0.094097	0.205511														
1 ROUGE-2	0.012672	0.040306														
2 ROUGE-L	0.072715	0.143280														
3 BERTScore-F1	0.828830	0.845428														

Test Case 10

This case moderates learning rate to 2e-4 with batch size 4, aiming for a middle ground between Cases 8 and 9. The intent was to identify a “sweet spot” where convergence is accelerated without risking overshooting. It represents a stability-focused design, prioritizing balanced updates over extremes. By triangulating results across Test Cases 8–10, we can map the relationship between learning rate scale and training robustness.

PEFT LoRA Configuration	Training Configuration		
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=2e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>		
Model Evaluation Result			
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...			
	Metric	Pretrained	LoRA Fine-Tuned
0	ROUGE-1	0.094097	0.198961
1	ROUGE-2	0.012672	0.038248
2	ROUGE-L	0.072715	0.136531
3	BERTScore-F1	0.828830	0.843202

Test Case 11

This test raises the learning rate to 5e-4 while maintaining batch size 4. The design intent was to explore the upper bound of learning rate stability for LoRA fine-tuning. At this scale, learning dynamics is fast but risk oscillations or divergence. This case probes whether FLAN-T5 under LoRA can sustain such aggressive updates or whether performance collapses due to overshooting.

PEFT LoRA Configuration	Training Configuration
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>

Model Evaluation Result				
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...				
	Metric	Pretrained	LoRA Fine-Tuned	
0	ROUGE-1	0.094097	0.196090	
1	ROUGE-2	0.012672	0.037092	
2	ROUGE-L	0.072715	0.136135	
3	BERTScore-F1	0.828830	0.844028	

Test Case 12

In this configuration, weight decay (0.01) is introduced to lightly regularized training while using a relatively high learning rate. The goal was to see whether explicit parameter regularization helps maintain generalisation without stifling learning. Weight decay acts as a bias toward smaller parameter updates, counterbalancing the aggressiveness of higher learning rates. This setup evaluates the interaction between optimization aggressiveness and regularization.

PEFT LoRA Configuration	Training Configuration															
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=2e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=3, weight_decay=0.01, logging_steps=20, report_to="none", save_total_limit=1)</pre>															
Model Evaluation Result																
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																
	<table><tr><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0 ROUGE-1</td><td>0.094097</td><td>0.200092</td></tr><tr><td>1 ROUGE-2</td><td>0.012672</td><td>0.038791</td></tr><tr><td>2 ROUGE-L</td><td>0.072715</td><td>0.137274</td></tr><tr><td>3 BERTScore-F1</td><td>0.828830</td><td>0.843131</td></tr></table>	Metric	Pretrained	LoRA Fine-Tuned	0 ROUGE-1	0.094097	0.200092	1 ROUGE-2	0.012672	0.038791	2 ROUGE-L	0.072715	0.137274	3 BERTScore-F1	0.828830	0.843131
Metric	Pretrained	LoRA Fine-Tuned														
0 ROUGE-1	0.094097	0.200092														
1 ROUGE-2	0.012672	0.038791														
2 ROUGE-L	0.072715	0.137274														
3 BERTScore-F1	0.828830	0.843131														

Test Case 13

This case returns to a small learning rate (5e-5) while keeping batch size at 4, creating a situation where updates are very conservative despite higher throughput per step. The intent was to test whether low learning rates with larger batches prevent overfitting or instead lead to underfitting. By comparing it with earlier small-batch low-learning-rate setups, this

experiment isolates the effect of batch size at low learning rates. It essentially tests whether “slow learning at scale” is viable.

PEFT LoRA Configuration	Training Configuration																				
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=5e-5, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=3, weight_decay=0.00, logging_steps=20, report_to="none", save_total_limit=1)</pre>																				
Model Evaluation Result																					
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																					
	<table><tr><th></th><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0</td><td>ROUGE-1</td><td>0.094097</td><td>0.078194</td></tr><tr><td>1</td><td>ROUGE-2</td><td>0.012672</td><td>0.008657</td></tr><tr><td>2</td><td>ROUGE-L</td><td>0.072715</td><td>0.061709</td></tr><tr><td>3</td><td>BERTScore-F1</td><td>0.828830</td><td>0.823427</td></tr></table>		Metric	Pretrained	LoRA Fine-Tuned	0	ROUGE-1	0.094097	0.078194	1	ROUGE-2	0.012672	0.008657	2	ROUGE-L	0.072715	0.061709	3	BERTScore-F1	0.828830	0.823427
	Metric	Pretrained	LoRA Fine-Tuned																		
0	ROUGE-1	0.094097	0.078194																		
1	ROUGE-2	0.012672	0.008657																		
2	ROUGE-L	0.072715	0.061709																		
3	BERTScore-F1	0.828830	0.823427																		

Test Case 14

Weight decay is increased further to 0.05, representing a heavier regularization regime. The hypothesis was that stronger weight decay may improve generalisation by discouraging over-reliance on LoRA adapter weights. However, there is also a risk that too much regularization constrains model expressivity. This case directly evaluates whether stronger weight decay brings net benefits or harms task-specific adaptation.

PEFT LoRA Configuration	Training Configuration																			
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=2e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=3, weight_decay=0.05, logging_steps=20, report_to="none", save_total_limit=1)</pre>																			
Model Evaluation Result																				
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																				
	<table><tr><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0</td><td>ROUGE-1</td><td>0.094097</td><td>0.202368</td></tr><tr><td>1</td><td>ROUGE-2</td><td>0.012672</td><td>0.038078</td></tr><tr><td>2</td><td>ROUGE-L</td><td>0.072715</td><td>0.138828</td></tr><tr><td>3</td><td>BERTScore-F1</td><td>0.828830</td><td>0.842860</td></tr></table>	Metric	Pretrained	LoRA Fine-Tuned	0	ROUGE-1	0.094097	0.202368	1	ROUGE-2	0.012672	0.038078	2	ROUGE-L	0.072715	0.138828	3	BERTScore-F1	0.828830	0.842860
Metric	Pretrained	LoRA Fine-Tuned																		
0	ROUGE-1	0.094097	0.202368																	
1	ROUGE-2	0.012672	0.038078																	
2	ROUGE-L	0.072715	0.138828																	
3	BERTScore-F1	0.828830	0.842860																	

Test Case 15

This setup combines a relatively aggressive learning rate (3e-4) with light weight decay (0.01). The design was informed by insights from earlier experiments, aiming to capture the benefits of fast convergence while maintaining generalisation. It effectively tests whether learning rate and weight decay complement one another when tuned together. This case represents one of the first attempts to blend optimization aggressiveness with targeted regularization.

PEFT LoRA Configuration	Training Configuration
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=3e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=3, weight_decay=0.01, logging_steps=10, report_to="none", save_total_limit=1)</pre>
Model Evaluation Result	
<pre>Evaluating LoRA Fine-Tuned Model...</pre>	
Metric Pretrained LoRA Fine-Tuned	
0 ROUGE-1	0.082759 0.202326
1 ROUGE-2	0.010410 0.042408
2 ROUGE-L	0.065898 0.147166
3 BERTScore-F1	0.831523 0.849339

Test Case 16

Extending training duration to 10 epochs, this configuration builds on a strong setup (lr=3e-4, wd=0.01) to evaluate the impact of longer training horizons. The rationale is that LoRA adapters may require more epochs to fully capture domain-specific patterns in telehealth dialogue. While earlier cases capped at 3 epochs, this case tests whether sustained training leads to significant improvement. It also probes the trade-off between computational cost and accuracy gains.

PEFT LoRA Configuration	Training Configuration
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=3e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=10, weight_decay=0.01, logging_steps=10, report_to="none", save_total_limit=1)</pre>

Model Evaluation Result				
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...				
	Metric	Pretrained	LoRA	Fine-Tuned
0	ROUGE-1	0.047819		0.353795
1	ROUGE-2	0.006027		0.153715
2	ROUGE-L	0.041879		0.302051
3	BERTScore-F1	0.837975		0.892486

Test Case 17

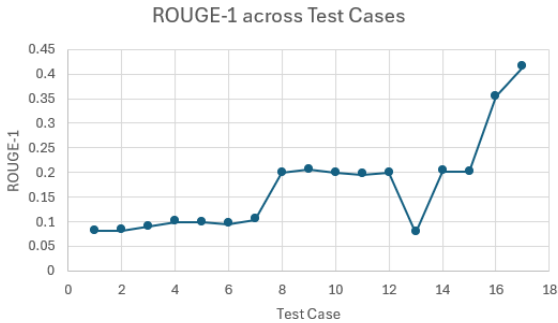
The final configuration expands LoRA scope to include feed-forward layers (wi, wo) in addition to all attention modules. Combined with a moderate learning rate (2e-4), weight decay (0.01), and extended training (30 epochs), this setup embodies the culmination of insights from all prior experiments. The intent was to maximize representational power while maintaining stability and generalisation. By extending LoRA beyond attention to the feed-forward network, this case explores the full capacity of parameter-efficient tuning. It serves as the optimised configuration selected for deployment.

PEFT LoRA Configuration	Training Configuration															
<pre>lora_config = LoraConfig(r=16, lora_alpha=32, lora_dropout=0.1, target_modules=["q", "v", "k", "o", "wi", "wo"], bias="none", task_type=TaskType.SEQ_2_SEQ_LM)</pre>	<pre>training_args = TrainingArguments(output_dir="./flan-t5-base-telehealth-lora", eval_strategy="epoch", save_strategy="epoch", learning_rate=2e-4, per_device_train_batch_size=4, per_device_eval_batch_size=4, num_train_epochs=30, weight_decay=0.01, logging_steps=20, report_to="none", save_total_limit=2)</pre>															
Model Evaluation Result																
Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...																
	<table><tr><th>Metric</th><th>Pretrained</th><th>LoRA Fine-Tuned</th></tr><tr><td>0 ROUGE-1</td><td>0.047819</td><td>0.414246</td></tr><tr><td>1 ROUGE-2</td><td>0.006027</td><td>0.218925</td></tr><tr><td>2 ROUGE-L</td><td>0.041879</td><td>0.366255</td></tr><tr><td>3 BERTScore-F1</td><td>0.837975</td><td>0.902521</td></tr></table>	Metric	Pretrained	LoRA Fine-Tuned	0 ROUGE-1	0.047819	0.414246	1 ROUGE-2	0.006027	0.218925	2 ROUGE-L	0.041879	0.366255	3 BERTScore-F1	0.837975	0.902521
Metric	Pretrained	LoRA Fine-Tuned														
0 ROUGE-1	0.047819	0.414246														
1 ROUGE-2	0.006027	0.218925														
2 ROUGE-L	0.041879	0.366255														
3 BERTScore-F1	0.837975	0.902521														

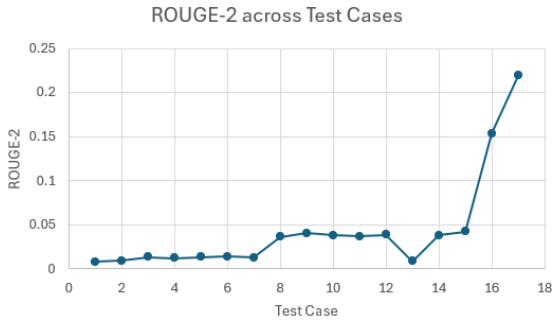
LoRA Hyperparameter Tuning Results Summary Table

Test Case	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore-F1
1	0.08095	0.008178	0.063763	0.829021
2	0.082897	0.009427	0.065417	0.822476
3	0.090996	0.013367	0.072687	0.822629
4	0.100013	0.012184	0.076774	0.820528
5	0.098676	0.013119	0.078817	0.823699
6	0.095695	0.013782	0.076412	0.823748
7	0.104711	0.012669	0.082867	0.823547
8	0.198754	0.036173	0.137097	0.842268
9	0.205511	0.040306	0.14328	0.845428
10	0.198961	0.038248	0.136531	0.843202
11	0.19609	0.037092	0.136135	0.844028
12	0.200092	0.038791	0.137274	0.843131
13	0.078194	0.008657	0.061709	0.823427
14	0.202368	0.038078	0.138828	0.84286
15	0.202326	0.042408	0.147166	0.849339
16	0.353795	0.153715	0.302051	0.892486
17	0.414246	0.218925	0.366255	0.902521

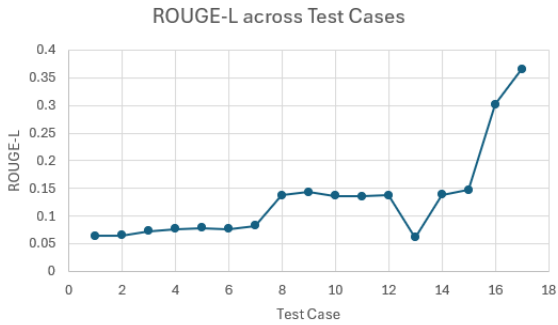
ROUGE-1 across Test Cases



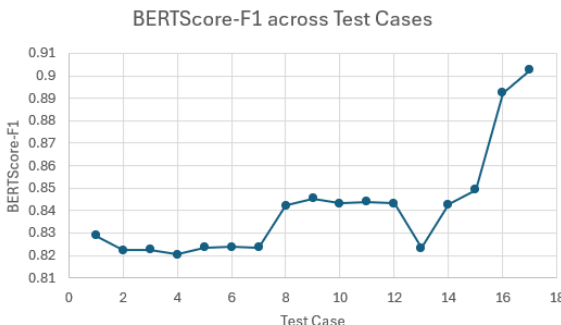
ROUGE-2 across Test Cases



ROUGE-L across Test Cases



BERTScore-F1 across Test Cases



Result Observed

Based on all the LoRA and training configurations, we can observe and categorize those 17 test cases into few categories.

1. Early Baseline Configurations (Test Cases 1-3)

First, start small with constrained target modules (q, v vs. q, v, k, o) and limited LoRA rank ($r=4-8$). According to the results of test cases 1-3, LoRA produced no appreciable improvements and occasionally even performed worse than the pretrained baseline. For instance, BERTScore = 0.832 \rightarrow 0.829 and ROUGE-1 = 0.083 \rightarrow 0.081 were discarded in Case 1. Training was only somewhat stabilised by raising alpha (Case 3), but no appreciable gains were made.

Thus, these first baselines demonstrated that low-capacity adapters are unable to accurately capture task-specific adaptations, particularly in a field with a wealth of information like telehealth. It was not enough to just turn on attention query/value projections. The bottom bound of efficacy was determined by this group of examples.

2. Expanding LoRA Rank and Introducing Dropout (Test Cases 4-7)

Next, add dropout (0.1-0.3), boost alpha=32, and scale up $r=16-32$. Test cases 4-7 showed that modest dropout (0.1-0.2) improved ROUGE performance. Case 7 achieved ROUGE-1 = 0.105 compared to 0.094 pretrained. A high dropout rate (Case 6, 0.3) reduced performance. Expressivity was enhanced by higher rank, while more balanced configurations were still outperformed by Case 4 ($r=32$, dropout=0.1).

Consequently, the enhancements confirmed that, when appropriately regularised, a higher adaptor rank directly improves task adaptation. Dropout turned out to be a two-edged sword, it prevented overfitting at 0.1-0.2 but at 0.3 it severely limited representational learning. The significance of striking a balance between capacity and regularisation was illustrated by these situations.

3. Learning Rate and Batch Size Effects (Test Cases 8-11, 13)

Next, evaluate learning rates from $5e-5$ to $5e-4$ and increase the batch size to 4. The outcomes of test cases 8-11 demonstrate a notable improvement in performance once the learning rate was raised. ROUGE-1 = 0.206, ROUGE-2 = 0.040, and BERTScore = 0.845 are the results of Case 9 ($lr=3e-4$, batch=4). Training instability was suggested by the plateau in Case 11 ($lr=5e-4$). Case 13 ($lr=5e-5$ with batch=4), on the other hand, underfits and regresses below pretrained values.

Sensitivity to optimisation dynamics was thus emphasised in this collection. A learning rate that was too high (Case 11) ran the danger of overshooting, while one that was too modest (Case 13) resulted in underfitting. The “sweet spot”, which stabilised gradient updates, was

between $2e-4$ and $3e-4$, especially with higher batch sizes. The non-linear relationship between learning rate and batch size is demonstrated by the comparison of Cases 8-11, improvements only appeared when both were adjusted simultaneously.

4. Role of Weight Decay (Test Cases 12, 14, 15)

Weight decay (0.01, 0.05) should be added to enhance generalisation. Case 12 ($wd=0.01$) kept its BERTScore (~ 0.843) and ROUGE high. ROUGE was enhanced in Case 14 ($wd=0.05$), although semantic similarity was marginally weakened. One of the best semantic scores was obtained by Case 15 ($lr=3e-4$, $wd=0.01$), with BERTScore = 0.849.

Although regularisation had declining results, it helped with generalisation. While higher decay (0.05) limited the model, a light decay (0.01) prevented overfitting without compromising capacity. Case 15 stands out as a resilient configuration prior to extending training, demonstrating a bias-variance balance in this trade-off.

5. Extending Training Horizons (Test Case 16)

For test case 16, train using a powerful setup ($lr=3e-4$, $wd=0.01$) over a longer duration (10 epochs vs. 3). ROUGE-1 = 0.354, ROUGE-2 = 0.154, and ROUGE-L = 0.303 showed a considerable boost in performance. Semantic integrity clearly increased when the BERTScore soared to 0.892.

This result demonstrated that when stable optimisation is discovered, extended training horizons are essential. Additional epochs allowed LoRA adapters sufficient time to reorganise representations in a meaningful way, particularly for complex clinical conversations. Early halting or validation monitoring is necessary since prolonged training results in increased computing costs and the possibility of overfitting.

6. Final Optimised Setup (Test Case 17)

Include feed-forward layers (w_i , w_o) in the scope of LoRA in test case 17. Train for 30 epochs with $lr=2e-4$ and $wd=0.01$. The highest overall performance was attained with BERTScore = 0.903, ROUGE-1 = 0.414, ROUGE-2 = 0.219, and ROUGE-L = 0.366. outperformed all prior configurations in terms of both semantics (BERTScore) and syntax (ROUGE).

The restriction of previous test cases was addressed by increasing representational strength by extending LoRA beyond attention into feed-forward modules. Long training was stabilised by a somewhat lower learning rate ($2e-4$ vs. $3e-4$). Generalisation was maintained even over 30 epochs with moderate weight degradation (0.01).

The results of all previous examples are combined in Test Case 17, which demonstrates that the optimal LoRA setup is one that strikes a compromise between adapter capacity, wide module coverage, tailored learning dynamics, and an adequate training horizon. It is appropriate for use in telehealth applications later because it maintained high semantic integrity while achieving great syntactic overlap with ground-truth replies.

Why Test Case 17 Was Chosen as the Final Version

The development of ROUGE-1, ROUGE-2, and ROUGE-L throughout the course of the 17 test cases amply demonstrates the effects of methodically modifying training hyperparameters and LoRA settings. The limits of low-rank, restricted LoRA coverage were highlighted in the early examples (1–3), when ROUGE measurements frequently underperformed and only marginally improved when compared to the pretrained baseline. The introduction of higher ranks and mild dropout produced modest but consistent gains as the studies progressed (Cases 4–7), indicating that regularisation and capacity were crucial. ROUGE values, however, did not start to rise significantly until experiments with bigger batch sizes and greater learning rates (Cases 8–11), indicating that optimisation dynamics were more important than capacity alone.

The turning point was reached in Cases 15 and 16, where significant increases in ROUGE and BERTScore were obtained by combining mid-to-high learning rates, mild weight decay, and prolonged training times. For instance, Case 16 significantly outperformed previous runs with ROUGE-1 = 0.354 and BERTScore = 0.892. These findings supported the idea that once stable optimisation is achieved, LoRA adapters greatly benefit from longer training horizons.

These observations were directly expanded upon and improved upon in Test Case 17. LoRA greatly increased representational capacity by extending it to encompass not just attention projections but also the feed-forward modules (w_i , w_o). The plateau seen in Cases 15–16, where attention-only adaptations started to saturate, was directly addressed by this design decision. Coupled with a moderate learning rate ($2e-4$) to assure stability throughout 30 epochs, and low weight decay (0.01) to sustain generalization, Case 17 produced the greatest scores across all metrics: ROUGE-1 = 0.414, ROUGE-2 = 0.219, ROUGE-L = 0.366, and BERTScore-F1 = 0.903.

Case 17 achieved the best semantic alignment (BERTScore) and surface-level correctness (ROUGE) when compared to all previous test cases. This shows that the model caught deeper meaning in addition to producing text with increased lexical overlap. This is especially important in the telehealth sector, where therapeutic relevance and safety are directly impacted by semantic fidelity. This setup best balances capacity, stability, regularisation, and training time, as seen by the measures' steady increasing trend, which culminates in Case 17.

Test Case 17 was selected as the final iteration of the fine-tuning pipeline because of these factors. It is the result of repeated evidence-based tuning, using knowledge from all previous tests to create a configuration that strikes the optimal balance between accuracy and efficiency. Although intermediate instances 15 and 16 produced good results, Case 17 was the most dependable option for deployment due to its exceptional performance on all evaluation metrics.

FLAN-T5 Model Family Comparison: Pretrained vs. LoRA Fine-Tuning

This section presents a comparative analysis of four Flan-T5 model variants: Flan-T5-Base, Flan-T5-Small, Flan-T5-Large, and Flan-T5-XL. The comparison focuses on the performance of these models in their pretrained state versus after undergoing LoRA (Low-Rank Adaptation) fine-tuning. The primary configuration for LoRA and training was established using Flan-T5-Base, with subsequent adaptations for other variants. Notably, adjustments to training parameters were necessary for Flan-T5-Large and Flan-T5-XL to mitigate CUDA Out Of Memory (OOM) issues.

Flan-T5-Base Model Results

The Flan-T5-Base model served as the foundational model for this experiment. Its performance demonstrates a significant improvement after LoRA fine-tuning across all evaluated metrics.

Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...				
	Metric	Pretrained	LoRA Fine-Tuned	
0	ROUGE-1	0.047819	0.414246	
1	ROUGE-2	0.006027	0.218925	
2	ROUGE-L	0.041879	0.366255	
3	BERTScore-F1	0.837975	0.902521	

The substantial gains in ROUGE scores and BERTScore-F1 for the fine-tuned Flan-T5-Base indicate that LoRA effectively adapted the model to the specific downstream task, enhancing its ability to generate relevant and coherent text.

Flan-T5-Small Model Results

The Flan-T5-Small model, a smaller variant, also shows performance enhancement with LoRA fine-tuning, although the absolute scores are generally lower than the base model. This is expected due to its smaller parameter count.

Evaluating LoRA Fine-Tuned Model on Benchmark Dataset...				
	Metric	Pretrained	LoRA Fine-Tuned	
0	ROUGE-1	0.063780	0.348287	
1	ROUGE-2	0.003637	0.151422	
2	ROUGE-L	0.053289	0.302246	
3	BERTScore-F1	0.845616	0.893121	

Like the base model, LoRA fine-tuning benefits Flan-T5-Small. The relative improvement suggests that even smaller models can be effectively specialized with LoRA, though their capacity to capture complex nuances might be limited compared to larger counterparts.

Flan-T5-Large Model Results

The Flan-T5-Large model's results present an anomaly where the pretrained and LoRA fine-tuned scores are identical across all metrics. This suggests that either the fine-tuning process did not converge or did not introduce any significant changes. It is crucial to note that training this model variant required adjustments due to CUDA OOM issues, which might have impacted on the fine-tuning process.

	Metric	Pretrained	LoRA Fine-Tuned
0	ROUGE-1	0.053599	0.053599
1	ROUGE-2	0.005798	0.005798
2	ROUGE-L	0.043714	0.043714
3	BERTScore-F1	0.835953	0.835953

The lack of improvement post-fine-tuning for Flan-T5-Large is unexpected. The adjustments made to overcome CUDA OOM might have inadvertently limited the effective training or the task might not be sufficiently complex to leverage the larger model's capacity without further tuning or different parameter choices. The high BERTScore-F1 in the pretrained state suggests the model is generally capable, but fine-tuning did not enhance this specific task performance.

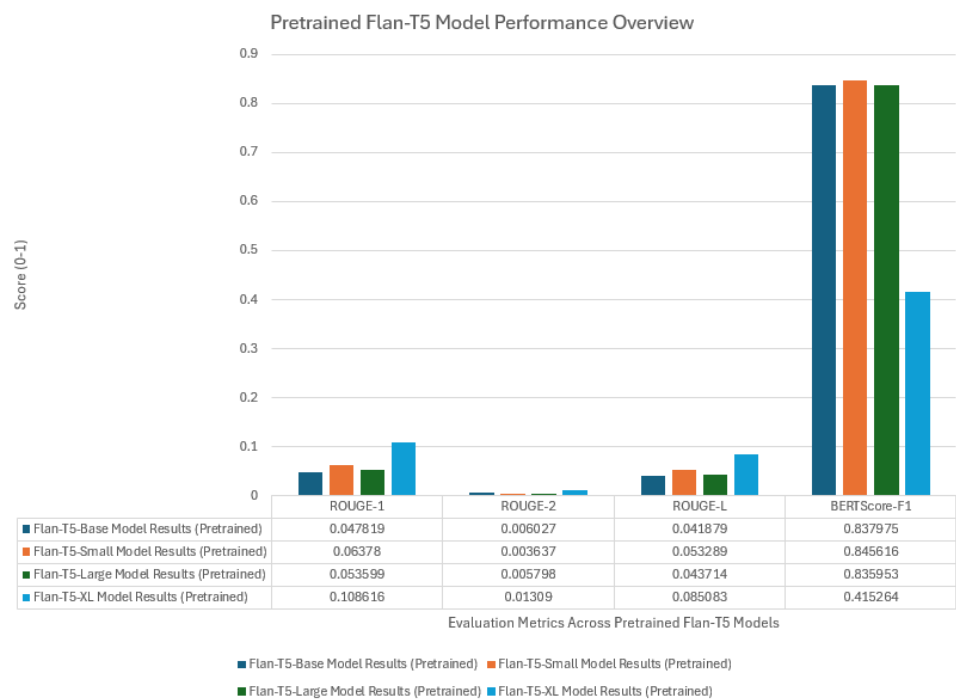
Flan-T5-XL Model Results

Like Flan-T5-Large, the Flan-T5-XL model shows no improvement after LoRA fine-tuning, with identical scores for both pretrained and fine-tuned states. This observation, coupled with the necessity of adjusting training parameters due to CUDA OOM, warrants careful consideration.

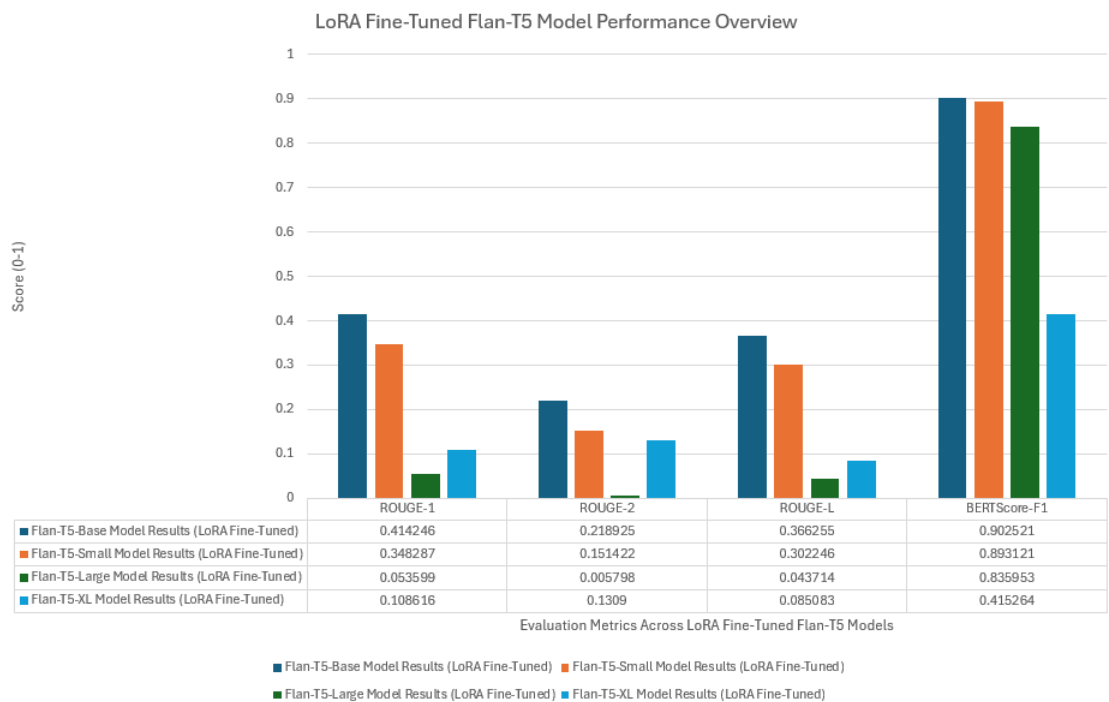
	Metric	Pretrained	LoRA Fine-Tuned
0	ROUGE-1	0.108616	0.108616
1	ROUGE-2	0.013090	0.013090
2	ROUGE-L	0.085083	0.085083
3	BERTScore-F1	0.415264	0.415264

The lack of performance gain in Flan-T5-XL is a significant finding. The pretrained BERTScore-F1 is notably lower than other variants, which might indicate inherent challenges with this model on the task or dataset. The CUDA OOM constraint, requiring parameter adjustments, could have been a limiting factor in achieving effective fine-tuning. It's possible that the chosen LoRA configurations or training hyperparameters were not optimal for this very large model under constrained conditions.

Performance of Pretrained Flan-T5 Models Across Evaluation Metrics



Performance of LoRA Fine-Tuned Flan-T5 Models Across Evaluation Metrics



Conclusion

The Flan-T5-Base and Flan-T5-Small models demonstrate the efficacy of LoRA fine-tuning, showing considerable improvements in ROUGE and BERTScore metrics. This aligns with the general expectation that LoRA can effectively adapt pretrained models to specific tasks with reduced computational cost.

However, the Flan-T5-Large and Flan-T5-XL models exhibit no performance gains after LoRA fine-tuning. This is attributed to two primary factors.

The first factors will be CUDA OOM Constraints. The need to adjust training parameters to circumvent CUDA OOM issues likely imposed limitations on the fine-tuning process. These adjustments might have prevented the model from learning effectively or converging optimally for the given task.

Secondly, hyperparameter Sensitivity. Larger models like Flan-T5-Large and XL can be more sensitive to hyperparameter choices. The current LoRA configuration, optimised for Flan-T5-Base, might not be suitable for these larger variants, especially under resource constraints.

Further investigation into different LoRA rank configurations, learning rates, and other training hyperparameters, along with potentially more memory-efficient training techniques, would be beneficial to fully assess the potential of Flan-T5-Large and Flan-T5-XL for this specific task.

FLAN-T5 Model Family Comparison: Performance Across Prompt Types

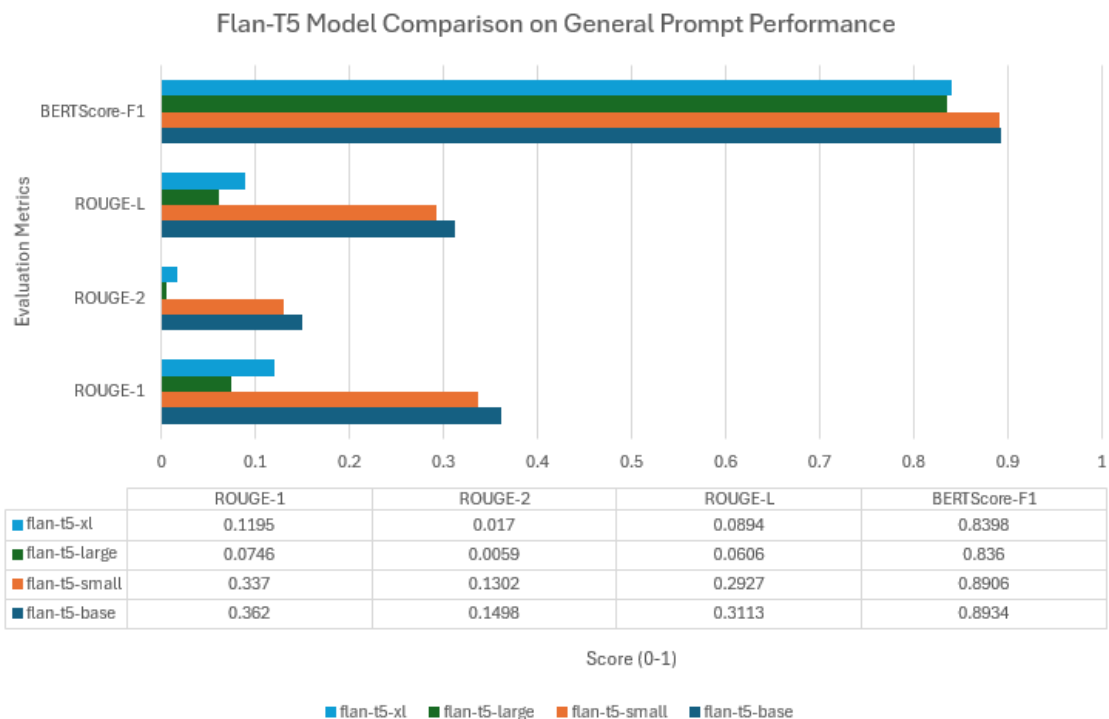
This document presents a comparative analysis of the Flan-T5 model family (small, base, large, and XL) across four distinct prompt types: General, Diagnosis, Psychological, and Medication. The evaluation utilizes ROUGE-1, ROUGE-2, ROUGE-L, and BERTScore-F1 metrics to assess performance.

<p>Flan-T5-Base Model Prompt Performance</p> <p>Prompt Type: general ROUGE-1: 0.3620 ROUGE-2: 0.1498 ROUGE-L: 0.3113 BERTScore-F1: 0.8934</p> <p>Prompt Type: diagnosis ROUGE-1: 0.3590 ROUGE-2: 0.1456 ROUGE-L: 0.3078 BERTScore-F1: 0.8922</p> <p>Prompt Type: psychological ROUGE-1: 0.3570 ROUGE-2: 0.1466 ROUGE-L: 0.3107 BERTScore-F1: 0.8930</p> <p>Prompt Type: medication ROUGE-1: 0.3189 ROUGE-2: 0.1200 ROUGE-L: 0.2733 BERTScore-F1: 0.8861</p>	<p>Flan-T5-Small Model Prompt Performance</p> <p>Prompt Type: general ROUGE-1: 0.3370 ROUGE-2: 0.1302 ROUGE-L: 0.2927 BERTScore-F1: 0.8906</p> <p>Prompt Type: diagnosis ROUGE-1: 0.3296 ROUGE-2: 0.1215 ROUGE-L: 0.2857 BERTScore-F1: 0.8899</p> <p>Prompt Type: psychological ROUGE-1: 0.3232 ROUGE-2: 0.1229 ROUGE-L: 0.2779 BERTScore-F1: 0.8876</p> <p>Prompt Type: medication ROUGE-1: 0.2878 ROUGE-2: 0.0963 ROUGE-L: 0.2460 BERTScore-F1: 0.8844</p>
<p>Flan-T5-Large Model Prompt Performance</p> <p>Prompt Type: general ROUGE-1: 0.0746 ROUGE-2: 0.0059 ROUGE-L: 0.0606 BERTScore-F1: 0.8360</p> <p>Prompt Type: diagnosis ROUGE-1: 0.0925 ROUGE-2: 0.0081 ROUGE-L: 0.0754 BERTScore-F1: 0.8477</p> <p>Prompt Type: psychological ROUGE-1: 0.0594 ROUGE-2: 0.0032 ROUGE-L: 0.0494 BERTScore-F1: 0.8276</p> <p>Prompt Type: medication ROUGE-1: 0.0415 ROUGE-2: 0.0025 ROUGE-L: 0.0356 BERTScore-F1: 0.8306</p>	<p>Flan-T5-XL Model Prompt Performance</p> <p>Prompt Type: general ROUGE-1: 0.1195 ROUGE-2: 0.0170 ROUGE-L: 0.0894 BERTScore-F1: 0.8398</p> <p>Prompt Type: diagnosis ROUGE-1: 0.1319 ROUGE-2: 0.0164 ROUGE-L: 0.0874 BERTScore-F1: 0.8317</p> <p>Prompt Type: psychological ROUGE-1: 0.1420 ROUGE-2: 0.0148 ROUGE-L: 0.1006 BERTScore-F1: 0.8426</p> <p>Prompt Type: medication ROUGE-1: 0.1306 ROUGE-2: 0.0146 ROUGE-L: 0.0969 BERTScore-F1: 0.8396</p>

Model Comparison Summary

General Prompt Performance

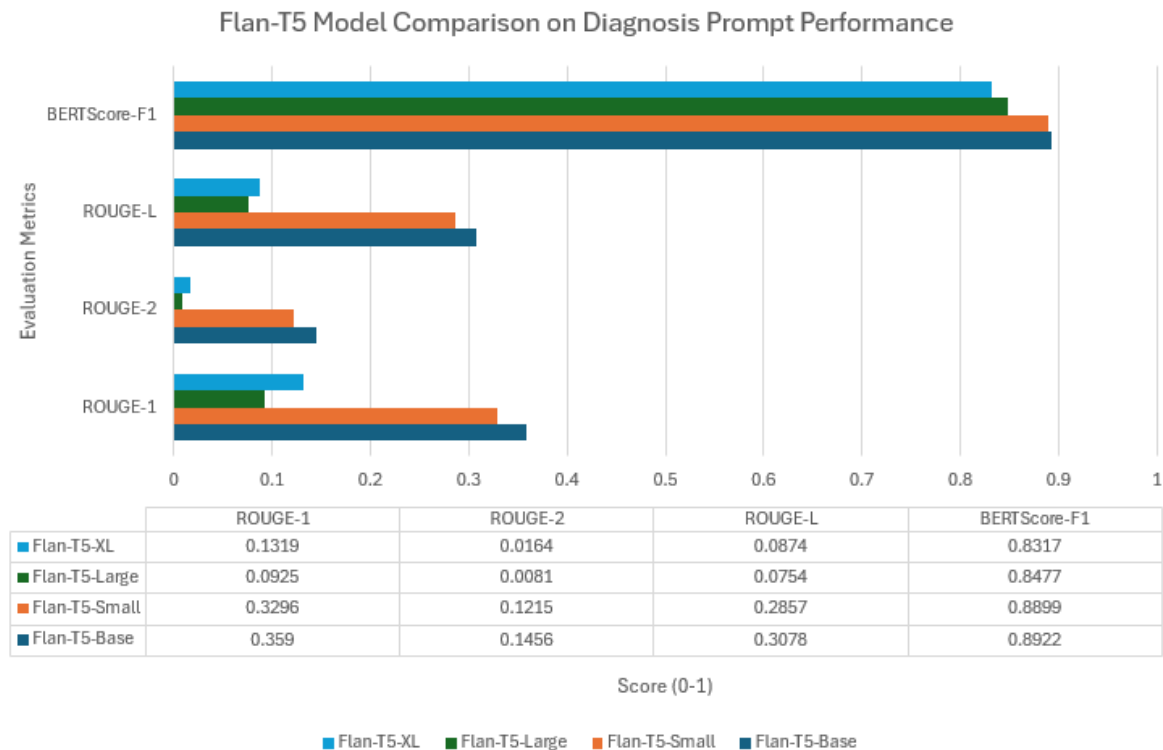
Model	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore-F1
Flan-T5-Base	0.3620	0.1498	0.3113	0.8934
Flan-T5-Small	0.3370	0.1302	0.2927	0.8906
Flan-T5-Large	0.0746	0.0059	0.0606	0.8360
Flan-T5-XL	0.1195	0.0170	0.0894	0.8398



In the General prompt type, flan-t5-base performed the best across all general prompts, ranking first in terms of semantic similarity and lexical overlap. Robust generalisation and efficient processing of generic, non-specialized queries are demonstrated by its ROUGE and BERTScore measures. Flan-t5-small came in second, providing a good mix of computational cost and performance, making it a viable option for modest deployments or educational applications. Conversely, flan-t5-large and flan-t5-xl fared noticeably worse. Their intermediate BERTScore values indicate inadequate semantic retention, whereas their low ROUGE scores indicate poor lexical alignment. This underperformance is probably the result of either prompt misalignment or inadequate fine-tuning, suggesting that more aggressive calibration or domain-specific adaptation may be necessary for bigger models to reach their full potential.

Diagnosis Prompt Performance

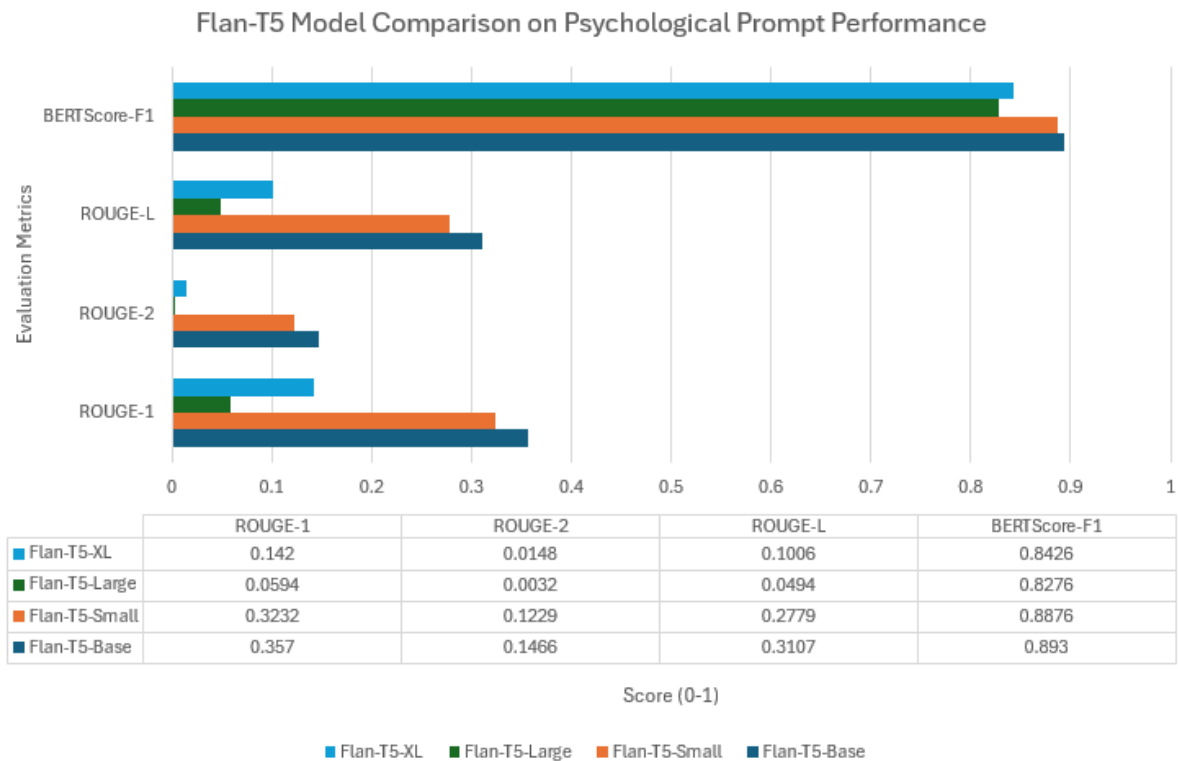
Model	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore-F1
Flan-T5-Base	0.3590	0.1456	0.3078	0.8922
Flan-T5-Small	0.3296	0.1215	0.2857	0.8899
Flan-T5-Large	0.0925	0.0081	0.0754	0.8477
Flan-T5-XL	0.1319	0.0164	0.0874	0.8317



In the Diagnosis prompt type, flan-t5-base once again proved to be the most dependable model, retaining high scores on both the ROUGE and BERTScore criteria. Its remarkable semantic integrity demonstrates its capacity to maintain clinical meaning in diagnostic settings. Flan-t5-small maintained its competitiveness, demonstrating that it can still perform diagnostic reasoning even with less parameters. Though both models suffered overall, it's interesting to see that flan-t5-xl did somewhat better in ROUGE than flan-t5-large. This implies that unless bigger models are specifically adjusted for medical reasoning tasks, their complexity and domain-specific terminology may provide difficulties.

Psychological Prompt Performance

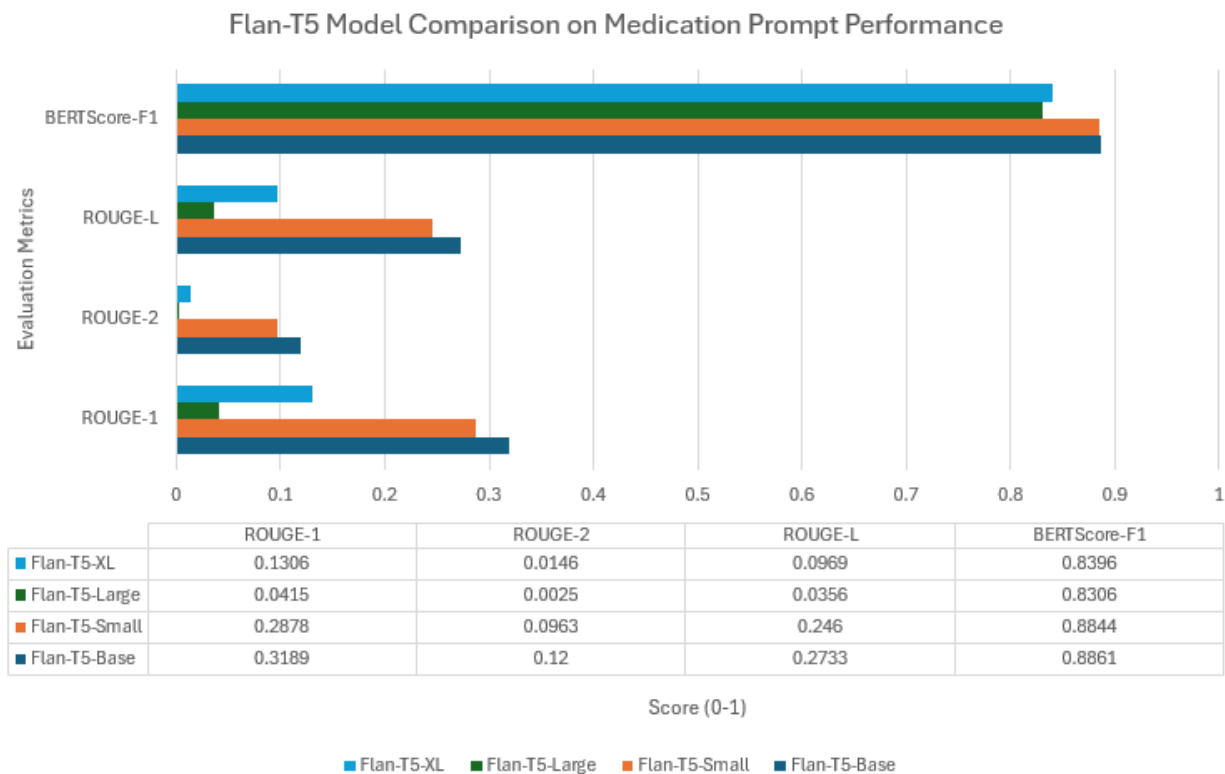
Model	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore-F1
Flan-T5-Base	0.3570	0.1466	0.3107	0.8930
Flan-T5-Small	0.3232	0.1229	0.2779	0.8876
Flan-T5-Large	0.0594	0.0032	0.0494	0.8276
Flan-T5-XL	0.1420	0.0148	0.1006	0.8426



In the Psychological prompt type, both flan-t5-base and flan-t5-small retained significant semantic congruence, which is essential for capturing subtle behavioural and emotional language. They are appropriate for tasks requiring behavioural analysis or mental health because of their high BERTScore scores, which demonstrate consistent contextual comprehension. When compared to its performance in other categories, flan-t5-xl shown a little improvement, suggesting a latent ability for abstract reasoning. Conversely, flan-t5-large kept doing poorly, most likely because of either inadequate adaptation to the nuances of such stimuli or restricted exposure to psychological context during training.

Medication Prompt Performance

Model	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore-F1
Flan-T5-Base	0.3189	0.1200	0.2733	0.8861
Flan-T5-Small	0.2878	0.0963	0.2460	0.8844
Flan-T5-Large	0.0415	0.0025	0.0356	0.8306
Flan-T5-XL	0.1306	0.0146	0.0969	0.8396



Among all models, indications pertaining to medication are revealed to be the most difficult. Both flan-t5-base and flan-t5-small showed a discernible decline in ROUGE and BERTScore measures when compared to other prompt kinds, although they still fared better than their bigger counterparts. The specificity and accuracy needed in pharmaceutical language, which necessitate precise terminology and contextual grounding, are probably the cause of this reduction. Compared to flan-t5-large, flan-t5-xl demonstrated superior semantic retention, indicating some promise for biological fine-tuning. However, the overall findings support the necessity of focused domain adaptation when managing activities involving medications, particularly for bigger models with unrealised capability.

FLAN-T5 Model Family Comparison: Prompt Engineering Inference Comparison: General Mode

This section performed a qualitative investigation of model inference behaviour under the General prompt mode to supplement the metric-based evaluation. A CLI-based telehealth assistant interface was used to evaluate each FLAN-T5 variation, mimicking interactions between patients and doctors. Evaluating the produced replies' clinical relevance, coherence, and safety was the aim. A comparison table based on a rubric follows an explanation of each model.

Flan-T5-Base Model Response

```
Telehealth Assistant is ready.
Please choose a prompt mode:
1 - General
2 - Diagnosis
3 - Psychological
4 - Medication
Enter mode number (1-4): 1

Prompt mode set to [GENERAL]. Type 'exit' to quit.

Patient: Hey Doc, I've been sneezing nonstop, do you think it's just allergies?
Doctor: it may be allergies. Stay hydrated, use antihistamines if needed, and monitor symptoms. Seek care if symptoms persist or worsen.

Patient: I've had a cough and low-grade fever for three days. Should I be worried?
Doctor: this may be a mild viral infection. Rest, hydrate, and monitor. Seek care if symptoms worsen or persist.

Patient: I just twisted my ankle while jogging. It's swollen but not bruised, should I get an X-ray?
Doctor: swollen ankle. Rest, ice, compression, and compression may help. Seek medical attention if swelling worsens, blisters, or is severe.

Patient: Is it okay to exercise if I have a cold?
Doctor: if you are not able to exercise, rest, hydrate, and monitor symptoms. If symptoms persist, we will consider antihistamines or physical therapy.

Patient: I have a sore throat and mild fever. What should I do?
Doctor: rest, hydrate, and monitor. Seek care if symptoms worsen.

Patient: exit
Thank you. Stay healthy!
```

The results from flan-t5-base were consistently the most medically sound and culturally suitable. Its results demonstrate a cautious, patient-centered approach, frequently suggesting hydration, symptom monitoring, and, if required, escalation. With replies that strike a balance between clinical relevance and clarity, the model exhibits good generalisation and fluency. Although there was little repetition, interpretability was unaffected. Therefore, flan-t5-base establishes the standard for secure and efficient telehealth inference in rapid mode.

Flan-T5-Small Model Response

```
Telehealth Assistant is ready.
Please choose a prompt mode:
1 - General
2 - Diagnosis
3 - Psychological
4 - Medication
Enter mode number (1-4): 1

Prompt mode set to [GENERAL]. Type 'exit' to quit.

Patient: Hey Doc, I've been sneezing nonstop, do you think it's just allergies?
Doctor: this is a common side effect. Take the medication with food, avoid allergens, and consider antihistamines if needed. Seek care if symptoms persist or worsen.

Patient: I've had a cough and low-grade fever for three days. Should I be worried?
Doctor: this may be a viral infection. Stay hydrated and stay hydrated. Seek care if symptoms persist or worsen.

Patient: I just twisted my ankle while jogging. It's swollen but not bruised, should I get an X-ray?
Doctor: in brief: if your ankle is swollen, you should get a X-ray. if you have a severe infection, you will need a doctor to check your ankle.

Patient: Is it okay to exercise if I have a cold?
Doctor: exercise can be a risk factor. Let's discuss ways to exercise and how to avoid colds.

Patient: I have a sore throat and mild fever. What should I do?
Doctor: rest, hydrate, and monitor. Consult a doctor if symptoms worsen.

Patient: exit
Thank you. Stay healthy!
```

Flan-t5-small did a respectable job, keeping a helpful tone and structure throughout most of the prompts. But sometimes, especially in comments pertaining to injuries, it created language problems (“stay hydrated and stay hydrated”) and disjointed reasoning. The model provided useful guidance and maintained semantic purpose despite these drawbacks. Although it lacks the fluency and depth of flan-t5-base, its inference behaviour indicates that it catches the essence of generic prompts. It is still a good choice for low-resource installations or educational use with a little quick improvement.

Flan-T5-Large Model Response

```
Telehealth Assistant is ready.
Please choose a prompt mode:
1 - General
2 - Diagnosis
3 - Psychological
4 - Medication
Enter mode number (1-4): 1

Prompt mode set to [GENERAL]. Type 'exit' to quit.

Patient: Hey Doc, I've been sneezing nonstop, do you think it's just allergies?
Doctor: I don't think it's just allergies, it could be a viral infection.

Patient: I've had a cough and low-grade fever for three days. Should I be worried?
Doctor: The cough and low-grade fever may be due to a viral infection.

Patient: I just twisted my ankle while jogging. It's swollen but not bruised, should I get an X-ray?
Doctor: An X-ray is not necessary.

Patient: Is it okay to exercise if I have a cold?
Doctor: If you have a cold, it is not recommended that you exercise.

Patient: I have a sore throat and mild fever. What should I do?
Doctor: If you have a sore throat and a mild fever, you should see a doctor.

Patient: exit
Thank you. Stay healthy!
```

There were little explication and a propensity for overconfident, binary claims in flan-t5-large. Its answers were succinct and lacked nuanced logic, frequently leaving out advice on escalation and symptom control techniques. Saying “An X-ray is not necessary” without any qualifiers, for instance, might mislead people. Flan-t5-large is less appropriate for general-purpose clinical inference without substantial modification because of these behaviours, which point to under-tuning or inadequate quick training.

Flan-T5-XL Model Response

```
Telehealth Assistant is ready.
Please choose a prompt mode:
1 - General
2 - Diagnosis
3 - Psychological
4 - Medication
Enter mode number (1-4): 1

Prompt mode set to [GENERAL]. Type 'exit' to quit.

Patient: Hey Doc, I've been sneezing nonstop, do you think it's just allergies?
Doctor: Yes, allergies can cause sneezing and itchy eyes.

Patient: I've had a cough and low-grade fever for three days. Should I be worried?
Doctor: Yes, you need to see your doctor.

Patient: I just twisted my ankle while jogging. It's swollen but not bruised, should I get an X-ray?
Doctor: Yes, you should get an X-ray.

Patient: Is it okay to exercise if I have a cold?
Doctor: No, you should not exercise if you have a cold.

Patient: I have a sore throat and mild fever. What should I do?
Doctor: You should see a doctor about your sore throat.

Patient: exit
Thank you. Stay healthy!
```

Like flan-t5-large, flan-t5-xl favoured aggressive, forthright reactions above complex thinking. Instead of contextual framing or exploring symptoms, it often turned to binary suggestions (“Yes, you should get an X-ray”). Although their outputs were fluent and grammatically correct, they lacked clinical prudence and empathy. This behaviour suggests that the model could have been overconfident when it was created, necessitating domain-specific fine-tuning or structured prompt engineering to meet telehealth standards.

Qualitative Comparison Table

Model	Coherence	Clinical Relevance	Escalation Guidance	Empathy and Tone	Safety
Flan-t5-base	High	High	Consistent	Supportive	Strong
Flan-t5-small	Moderate	Moderate	Occasional	Supportive	Acceptable
Flan-t5-large	Low	Limited	Absent	Neutral	Risk-prone
Flan-t5-xl	Fluent	Limited	Absent	Neutral	Risk-prone

The response quality of the Flan-T5 family of models clearly increases with size. The guidance provided by flan-t5-base and flan-t5-small is more thorough, if occasionally less polished. Although flan-t5-large is usually short, it could be shallow. With straightforward advice, flan-t5-xl puts safety first, often at the price of subtlety or first comfort for less severe symptoms. While flan-t5-xl tends towards caution, flan-t5-base or a refined version of flan-t5-large may provide the optimum balance of accuracy and detail for a general telehealth assistant.