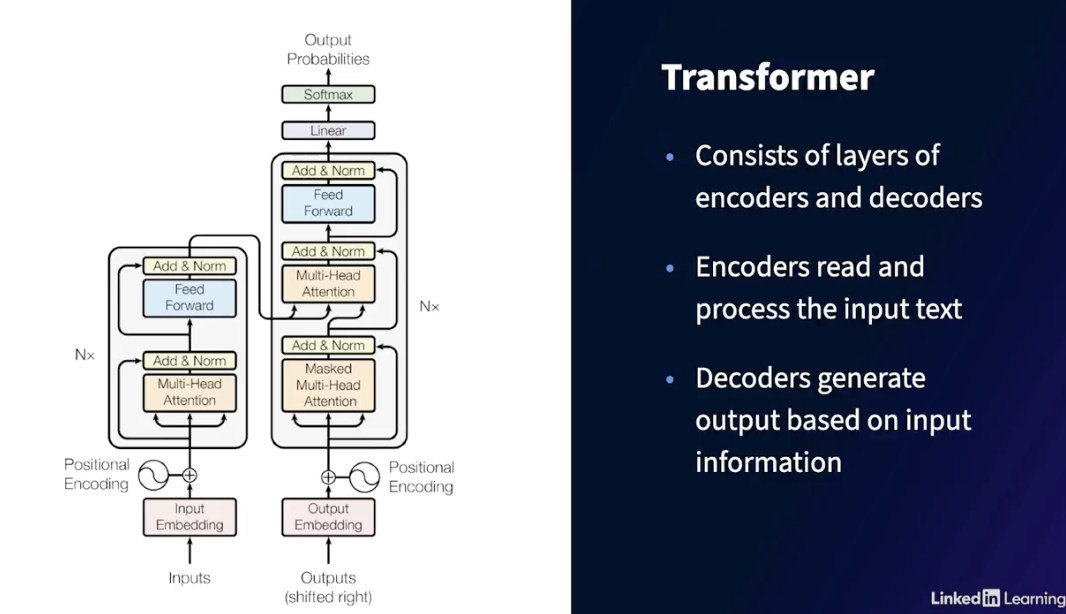
# Introduction to Large Language Models LLM

## Video: LLMs: Revolutionizing AI

* Impact on Daily Life: Large Language Models (LLMs) enhance interactions with technology, making them more natural and accessible, such as in virtual assistants like Siri and Alexa.
* Applications in Various Fields:  
  + Healthcare: LLMs assist doctors with quicker diagnostics and personalized treatment plans.
  + Education: They provide personalized tutoring, adapting to each student's learning style.
  + Business: LLMs streamline communication by drafting emails and generating reports, improving operational efficiency.
  + Creative Fields: They collaborate with artists to create novel artworks and write music, expanding creative possibilities.
* Role of LLMs: Large Language Models (LLMs) are compared to master chefs in a professional kitchen, where each tool and ingredient has a purpose. LLMs are central to technologies that understand and generate human-like text, enhancing interactions with machines.
* Impact on Daily Life:  
  + Virtual Assistants: LLMs power virtual assistants like Siri and Alexa, making interactions more natural and seamless.
  + Customer Support: They are used in customer support bots to assist users efficiently.
* Applications in Various Fields:
  + Healthcare: LLMs help doctors with quicker diagnostics and personalized treatment plans, acting like sous chefs that bring the best out of data.
  + Education: LLMs enable personalized tutoring, adapting to each student's learning style, providing a 24/7 tutor.
  + Business: They streamline communication by drafting emails and generating reports, functioning like an assistant who never sleeps and continuously learns to improve.
  + Creative Fields: LLMs collaborate with artists to create novel artworks and write music, pushing the boundaries of creativity.

## The architecture of LLMs



* Transformer Architecture: Large Language Models (LLMs) use transformers, which consist of layers of encoders and decoders. Encoders process the input text, while decoders generate the output.
* Self-Attention Mechanism: This mechanism allows the model to weigh the importance of different words in a sentence, improving its understanding of context and nuances.
* Parallel Processing: Unlike older models, transformers process all parts of the data simultaneously, significantly speeding up tasks.

## Ethical considerations surrounding LLMs

* Bias: LLMs can inadvertently learn and perpetuate biases present in their training data. Developers work to curate and diversify data sets and apply techniques to detect and mitigate bias.
* Transparency: It's crucial for developers to ensure transparency in how LLMs operate and make decisions, especially in sectors like healthcare or law, to build trust and accountability.
* Regulation: Ethical guidelines and regulations are needed to prevent misuse of LLMs and safeguard against unintended consequences, ensuring responsible use of the technology.

## Comparing LLMs

Types of LLM Architectures:

* + Encoder-only models (e.g., BERT): Focus on understanding and analyzing input data, suitable for tasks like sentiment analysis and question answering.
  + Decoder-only models (e.g., GPT series): Excel in generating text based on input, ideal for content creation tasks.
  + Encoder-decoder models (e.g., T5): Combine understanding and generating text, versatile for a range of applications like translation, classification, and summarization.

Model Selection:

* Choose based on the nature of the task: understanding, generating, or both.
* Consider computational resources and infrastructure capabilities, especially for large models.

Efficiency Techniques:

* Techniques like model distillation and quantization can reduce computational load, making it feasible to deploy powerful models in production environments.

## FLAN-T5 in focus

* Versatility: FLAN-T5 is a highly adaptable model that can handle a variety of NLP tasks like translation, summarization, and question answering by framing tasks as natural language instructions.
* Instruction Tuning: It enhances the original T5 model by using diverse prompts during training, allowing it to better understand and generate responses based on natural language instructions.
* Ease of Integration: FLAN-T5 can be easily integrated into applications using frameworks like the Hugging Face transformers library, making it accessible for developers to deploy powerful NLP tools quickly.

# Utilizing LLMs with Prompt Engineering

### Basics of prompt engineering

* Definition of a Prompt: A prompt is an instruction or set of instructions given to an LLM, setting the stage for how the model should respond.
* Importance of Context: Providing context helps the model understand not just what you're asking, but also how to tailor each response appropriately.
* Methods of Interaction: Interacting with LLMs can be done through applications, web interfaces, or directly through code, each offering different levels of control and flexibility.

### Crafting effective prompts

* Clarity and Specificity: A well-designed prompt should be clear, concise, and specific to guide the model's response effectively.
* Prompt Patterns: Three main patterns for prompt engineering are:  
  + Few-shot pattern: Providing several examples before presenting a new task.
  + Cognitive verifier pattern: Asking for additional information to ensure reliable outputs.
  + Question refinement pattern: Refining or clarifying a question before answering.

These patterns help maximize the efficiency and accuracy of interactions with LLMs.

### Prompt engineering with FLAN-T5

* Versatility of FLAN-T5: This model can handle tasks like text summarization, translation, and question answering using the Hugging Face transformers library and TensorFlow.
* Setup and Usage: The video covers setting up the environment, installing necessary libraries, and using the AutoTokenizer and TFAutoModelForSeq2SeqLM to process and generate text.
* Practical Examples: Demonstrations include summarizing text, translating text from English to French, and answering questions based on provided context.

These points highlight how to effectively utilize FLAN-T5 for various NLP tasks.

### Case studies in prompt engineering

* Customer Service: Companies like Zendesk and Salesforce use prompt engineering to make chatbots ask more specific questions, leading to more accurate and helpful responses, enhancing customer satisfaction.
* Healthcare: IBM's Watson improved its medical diagnosis capabilities by using detailed prompts to gather specific symptoms, medical history, and test results, resulting in more accurate insights.
* Media: Google News refined its prompts to extract key points, controversies, and implications from articles, leading to more accurate and informative news summaries.
* Language Learning: Duolingo increased user engagement by modifying prompts to create more engaging and relatable interactions, such as translating sentences as if speaking to a friend.

These examples demonstrate how precise and contextually appropriate prompts can significantly enhance the efficiency and effectiveness of AI applications across various industries.

## **Challenge: Design a Translation Prompt**

### **Objective:**

Craft a tailored prompt for translating dialogues using the few-shot learning pattern. Your goal is to enable effective and context-aware translations of dialogues from the DialogSum dataset, which contains summaries of dialogues.

### **Steps:**

1. Select dialogues: Choose three dialogues from the DialogSum dataset. Include the pre-existing summaries provided by the dataset as your baseline examples.
2. Create few-shot learning prompt: Construct a prompt that integrates the three chosen dialogues with their corresponding summaries. This prompt will serve as a model to teach the AI how to approach translating new dialogues while preserving the context highlighted in the summaries.
3. Specify the translation task: Clearly define the task within the prompt, stating that the objective is to translate the dialogue into another language, ensuring that the translation reflects the nuances and context captured in the English summary.
4. Implement and test with code: Use your coding skills to implement the prompt in a script. Utilize a platform like Hugging Face with a suitable model to translate a new dialogue from the dataset.
5. Evaluate and iterate: Analyze the effectiveness of the translation. Does it accurately reflect the provided summaries in the new language? Refine your prompt based on the results to improve accuracy and context preservation.

### **Challenge Guidelines:**

* Ensure that the AI’s translations maintain the integrity and context of the original dialogues as much as possible.
* Your prompt structure should facilitate clear understanding by the AI.
* Use precise and detailed instructions within your prompt to guide the AI's output effectively.

# Transfer Learning for LNP Tasks

## Transfer learning in LLMs

* Transfer Learning Analogy: Transfer learning is compared to a chef adapting skills to a new kitchen, where a pre-trained model is tweaked for a related task with minor modifications.
* Fine-Tuning Analogy: Fine-tuning is likened to a chef learning an entirely new cuisine, requiring extensive training on a new dataset, adjusting all model weights and biases.
* Application: Transfer learning involves adding a new component to a pre-trained model for specialized tasks, while fine-tuning adjusts the entire model for significantly different tasks.
* Resource Consideration: Transfer learning is quicker and less resource-intensive, ideal for similar tasks, whereas fine-tuning is more resource-intensive but necessary for very different tasks or when maximum accuracy is critical.

## Choosing models for transfer learning

* Criteria for Choosing Models: Consider the similarity of the source and target tasks, the size and quality of the pre-trained model, and compatibility with your specific requirements.
* Examples:
  + Healthcare: Using VGG-19 for pneumonia detection from chest X-rays due to its ability to transfer low-level features like edges and textures.
  + Sentiment Analysis: Leveraging BERT for analyzing product reviews, benefiting from its pre-trained language understanding.
  + Plant Disease Detection: Applying MobileNet for identifying plant diseases from images, optimized for resource-efficient applications.

These examples illustrate how transfer learning can be effectively applied across different domains by choosing the right model based on task similarity and dataset constraints.

## Evaluating transfer learning outcomes

* ROUGE and BLEU Metrics: These metrics are essential for assessing the quality of text generation tasks like summarization and translation.
* ROUGE: Measures the overlap between generated text and reference text, focusing on recall. For example, comparing how many key phrases in a generated summary match the reference summary. Measures overlap: *Number of unigrams overlapping / number of unigrams in original text*. Like **Recall**
* BLEU: Measures how well the generated text matches the reference text in terms of exact phrases and their order, focusing on precision. For example, comparing the sequence of words in a generated translation to the reference translation. *Number of unigrams overlapping / number of unigrams in generated text.* Like **Precision**

# PEFT fine-tuning with LoRA

## Introduction to PEFT

* Definition of PEFT: Parameter-efficient fine-tuning (PEFT) modifies a small subset of a model's parameters rather than the entire model, making it highly efficient.
* Comparison with Traditional Methods: Unlike traditional fine-tuning, which adjusts all parameters, and transfer learning, which adds new layers, PEFT uses lightweight modules called adapters.
* Advantages of PEFT: PEFT is particularly valuable when dealing with limited data, as it requires fewer resources and less computational power while still achieving high performance.

This approach is especially useful in scenarios where collecting large amounts of labeled data is impractical or too costly.

## LoRA adapters

* Definition and Function: LoRA (Low-Rank Adaptation) adapters are a type of parameter-efficient fine-tuning (PEFT) that focus on fine-tuning a small subset of parameters in pre-trained models.
* Efficiency: By updating only low-rank matrices (A and B), LoRA adapters significantly reduce the number of parameters to be trained, making the process faster and less resource-intensive.
* Application: LoRA adapters are particularly useful for adapting models to new tasks with limited data and computational resources, providing significant performance improvements without modifying the entire model.

## LoRA in depth: Technical analysis

* Balancing Overfitting and Generalizability: It's crucial to balance the model's performance to avoid overfitting while ensuring it generalizes well to new data.
* Rank Selection: Choosing the right rank for LoRA adapters is essential. A lower rank can help prevent overfitting but might limit learning capacity, while a higher rank enhances learning but increases the risk of overfitting.
* Parameter Tuning: Fine-tuning parameters like learning rate, batch size, and the number of epochs is important to optimize model training without overfitting.

# Creating a full NLP solution

## **Challenge: Finetuning the Sentiment Analysis Model**

Ready to elevate your machine learning expertise? In this challenge, you'll fine-tune a sentiment analysis model using DistilBERT and a sentiment analysis dataset. This exercise will empower you to enhance a pre-trained model's ability to accurately assess sentiment in text, a crucial skill in NLP applications.

#### Steps:

1. Load data:

* Download and preprocess a sentiment analysis dataset, such as the SST-2 dataset, to prepare it for training.

2. Initialize model:

* Load the pre-trained DistilBERT model and tokenizer from Hugging Face's Transformers library.

3. Prepare data for training:

* Tokenize the dataset and create training and validation splits.

4. Fine-tune the model:

* Train the DistilBERT model on the tokenized dataset, adjusting its parameters to learn sentiment classification.

5. Evaluate performance:

* Assess the model's performance using metrics such as accuracy and F1 score to ensure it accurately predicts sentiment.

#### Conclusion:

By completing this challenge, you've gained hands-on experience in fine-tuning a sentiment analysis model. This forms a vital component of a comprehensive NLP solution, where sentiment analysis, translation, and Q&A capabilities work together to provide powerful, integrated AI applications.

### Step-by-Step: Fine-Tuning the Sentiment Analysis Model

#### Introduction:

In this exercise, we'll fine-tune the DistilBERT model for sentiment analysis using the SST-2 dataset. We'll break down each step and explain the code snippets in detail to ensure you understand how to implement transfer learning using TFAutoModelForSequenceClassification.

#### Steps:

Load data:

* Download and preprocess the IMDb movie reviews dataset.

from datasets import load\_dataset

dataset = load\_dataset('stanfordnlp/sst2')

*Explanation:* This code uses the load\_dataset function from the datasets library to download the SST-2 dataset. The dataset is automatically split into training and testing sets.

Initialize model:

* Load the pre-trained DistilBERT model and tokenizer.

from transformers import TFAutoModelForSequenceClassification, AutoTokenizer

model\_name = "distilbert-base-uncased"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

model = TFAutoModelForSequenceClassification.from\_pretrained(model\_name, num\_labels=2)

*Explanation:*

* model\_name specifies the pre-trained model we want to use, distilbert-base-uncased.
* AutoTokenizer is used to preprocess the text data, converting it into a format that the model can understand.
* TFAutoModelForSequenceClassification loads the DistilBERT model tailored for sequence classification tasks. num\_labels=2 specifies that we have two output classes (positive and negative sentiment).

Prepare data for training:

* Tokenize the dataset and create training and validation splits.

def tokenize\_function(examples):

return tokenizer(examples['sentence'], truncation=True, padding=True)

tokenized\_datasets = dataset.map(tokenize\_function, batched=True)

train\_dataset = tokenized\_datasets["train"].to\_tf\_dataset(

columns=["input\_ids", "attention\_mask"],

label\_cols="label",

shuffle=True,

batch\_size=64

)

validation\_dataset = tokenized\_datasets["validation"].to\_tf\_dataset(

columns=["input\_ids", "attention\_mask"],

label\_cols="label",

shuffle=False,

batch\_size=64

)

*Explanation:*

* tokenize\_function applies the tokenizer to each text in the dataset, ensuring text is truncated to fit the model's input requirements and padded to the same length.
* dataset.map(tokenize\_function, batched=True) applies this function to the entire dataset in batches for efficiency.

Fine-tune the model:

* Compile and train the model.

from tensorflow.keras.optimizers import Adam

model.compile(

optimizer=tf.keras.optimizers.Adam(learning\_rate=5e-5),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=tf.metrics.SparseCategoricalAccuracy(),

)

model.fit(train\_dataset, epochs=3, validation\_data=test\_dataset)

*Explanation:*

* Adam(learning\_rate=5e-5) specifies the optimizer and learning rate for training.
* model.compile sets up the model with the Adam optimizer, a loss function suitable for classification, and sets accuracy as a metric to monitor.
* model.fit trains the model on the training dataset for three epochs and evaluates it on the test dataset after each epoch to monitor performance.

Evaluate Performance:

* Assess the model's performance.

loss, accuracy = model.evaluate(test\_dataset)

print(f"Test Accuracy: {accuracy:.2f}")

*Explanation:*

* model.evaluate(test\_dataset)calculates the loss and accuracy of the model on the test dataset.
* print(f"Test Accuracy: {accuracy:.2f}") outputs the accuracy, giving a clear metric of the model's performance on unseen data.

### 

#### Conclusion:

By following these steps, you've successfully fine-tuned a DistilBERT model for sentiment analysis using transfer learning. This approach leverages pre-trained knowledge to adapt the model efficiently to specific tasks. This skill is essential for creating comprehensive NLP solutions that integrate sentiment analysis, translation, and Q&A capabilities.

## **Challenge: Fine-Tuning the Q&A model**

#### Introduction:

Ready for an exciting challenge? This time, you'll fine-tune a Question & Answer (Q&A) model using the FLAN-T5-large model and the SQuAD dataset, leveraging transfer learning. This exercise will help you adapt a powerful pre-trained model to excel at answering questions based on given passages, an essential skill for building advanced NLP solutions.

#### Steps:

1. Load data:

* Download and preprocess the SQuAD dataset to prepare it for training.

2. Initialize model:

* Load the pre-trained FLAN-T5-large model and tokenizer from Hugging Face's Transformers library.

3. Do transfer learning:

* Freeze the model’s layers to enable efficient fine-tuning with limited data.

4. Prepare data for training:

* Tokenize the dataset and create training and validation splits.

5. Fine-tune the model:

* Train the FLAN-T5-large model on the tokenized dataset, adjusting only the parameters of the LoRA adapters.

6. Evaluate performance:

* Assess the model's performance using metrics such as BLUE or ROUGE score and Exact Match to ensure it accurately answers questions.

#### Conclusion:

By completing this challenge, you've learned to fine-tune a Q&A model using FLAN-T5-large, an essential component of a comprehensive NLP solution. This ties together with sentiment analysis and translation to create a robust system capable of understanding, interpreting, and responding to human language in diverse applications.

## **Step-by-Step: Fine-Tuning the Q&A Model**

### **Introduction:**

In this exercise, we'll fine-tune the FLAN-T5-large model for Question & Answer (Q&A) tasks using the SQuAD dataset and transfer learning. We'll break down each step and explain the code snippets in detail to ensure beginners understand how each part fits into the overall process.

### **Steps:**

Load data:

* Download and preprocess the SQuAD dataset.

from datasets import load\_dataset

dataset = load\_dataset('squad\_v2')

*Explanation:*

This code uses the load\_dataset function from the datasets library to download the SQuAD dataset, which is a popular dataset for Q&A tasks.

Initialize model:

* Load the pre-trained FLAN-T5-large model and tokenizer.

from transformers import TFAutoModelForSeq2SeqLM, AutoTokenizer

model\_name = "google/flan-t5-large"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

model = TFAutoModelForSeq2SeqLM.from\_pretrained(model\_name)

*Explanation:*

* model\_name specifies the pre-trained model we want to use, google/flan-t5-large.
* AutoTokenizer is used to preprocess the text data, converting it into a format that the model can understand.
* TFAutoModelForSeq2SeqLM loads the FLAN-T5-large model tailored for sequence-to-sequence learning tasks.

Prepare data for training:

* Tokenize the dataset and create training and validation splits.

# Tokenize the dataset["train"].select(range(25000))

def preprocess\_function(examples):

inputs = [context + " question: " + question for question, context in zip(examples["question"], examples["context"])]

targets = [answer['text'][0] if len(answer['text']) > 0 else "" for answer in examples["answers"]]

model\_inputs = tokenizer(inputs, max\_length=512, truncation=True, padding='max\_length')

# Setup the tokenizer for targets

with tokenizer.as\_target\_tokenizer():

labels = tokenizer(targets, max\_length=512, truncation=True, padding='max\_length')

model\_inputs["labels"] = labels["input\_ids"]

model\_inputs["decoder\_input\_ids"] = labels["input\_ids"]

return model\_inputs

train\_dataset = dataset["train"].select(range(25000)).map(preprocess\_function, batched=True)

validation\_dataset = dataset["validation"].select(range(2000)).map(preprocess\_function, batched=True)

*Explanation:*

* preprocess\_function tokenizes the questions and answers, truncating and padding them to fit the model's input requirements.
* tokenized\_datasets applies this function to the entire dataset in batches, making it ready for training.
* The data is then shuffled and batched for efficient training and validation.

Create TF datasets:

# Data collator

data\_collator = DataCollatorForSeq2Seq(tokenizer, model=None)

# Convert the tokenized dataset to a TensorFlow dataset

train\_dataset = train\_dataset.to\_tf\_dataset(

columns=["input\_ids", "attention\_mask", "decoder\_input\_ids"],

label\_cols="labels",

shuffle=True,

batch\_size=64,

collate\_fn=data\_collator

)

validation\_dataset = validation\_dataset.to\_tf\_dataset(

columns=["input\_ids", "attention\_mask", "decoder\_input\_ids"],

label\_cols="labels",

shuffle=False,

batch\_size=64,

collate\_fn=data\_collator

)

*Explanation:*

* Remember that in seq2seq models we need to add as columns the decoder\_input\_ids.
* We use a data\_collator to ensure the last batch (which is not 64 rows long) is dropped.

Fine-tune the model:

* Compile and train the model.

from tensorflow.keras.optimizers import Adam

model.compile(optimizer=Adam(learning\_rate=5e-5),

loss=model.compute\_loss)

model.fit(train\_dataset, epochs=3, validation\_data=validation\_dataset)

*Explanation:*

* Adam(learning\_rate=5e-5) specifies the optimizer and learning rate for training.
* model.compile sets up the model with the Adam optimizer, a loss function suitable for sequence-to-sequence learning, and sets accuracy as a metric to monitor.
* model.fit trains the model on the training dataset for three epochs and evaluates it on the validation dataset after each epoch to monitor performance.

Evaluate Performance:

* Assess the model's performance.Explanation:

rom rouge\_score import rouge\_scorer

from nltk.translate.bleu\_score import sentence\_bleu, SmoothingFunction

import nltk

nltk.download('punkt')

def answer(inputs):

outputs = model.generate(inputs[0]["input\_ids"], max\_length=128, num\_beams=4, early\_stopping=True)

return tokenizer.decode(outputs[0], skip\_special\_tokens=True)

# Function to calculate ROUGE and BLEU scores

def calculate\_scores(reference, hypothesis):

# Initialize scorers

rouge = rouge\_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use\_stemmer=True)

# Calculate ROUGE scores

rouge\_scores = rouge.score(reference, hypothesis)

return rouge\_scores

# Evaluate translations and calculate scores

batch = next(iter(validation\_dataset))

answer = answer(batch)

reference\_text = tokenizer.decode(batch[1][0], skip\_special\_tokens=True)

rouge\_scores = calculate\_scores(reference\_text, answer)

print(f"Reference: {reference\_text}")

print(f"Translation: {answer}")

print(f"ROUGE Scores: {rouge\_scores}")

print()

*Explanation:*

* We use the rouge\_scorer to calculate the Rouge scores.
* model.generate(validation\_dataset) generates predictions for the validation dataset.
* answer and references prepare the predictions and references in the required format for the evaluation metric.
* calculate\_scores calculates the performance metrics and prints the results.

### 

### **Conclusion:**

By following these steps, you've successfully fine-tuned the FLAN-T5-large model for Q&A. This approach leverages pre-trained knowledge and efficient fine-tuning techniques, allowing you to create robust NLP solutions that integrate sentiment analysis, translation, and Q&A capabilities.

## **Challenge: Fine-Tuning the Summarization Model**

### **Introduction:**

Ready for another challenge? This time, you'll fine-tune a summarization model using FLAN-T5 and the CNN/Daily Mail dataset, leveraging the efficiency of LoRA adapters. Summarization is a key NLP task, and this exercise will help you adapt a pre-trained model to generate concise summaries of long texts.

### **Steps:**

1. Load data:

* Download and preprocess the CNN/Daily Mail dataset to prepare it for training.

2. Initialize model:

* Load the pre-trained FLAN-t5 model and tokenizer from Hugging Face's Transformers library.

3. Integrate LoRA adapters:

* Add LoRA adapters to the multi-head attention components of the model’s layers to enable efficient fine-tuning with limited data.

4. Prepare data for training:

* Tokenize the dataset and create training and validation splits.

5. Fine-tune the model:

* Train the FLAN-t5 model on the tokenized dataset, adjusting only the parameters of the LoRA adapters.

6. Evaluate performance:

* Assess the model's performance using metrics such as ROUGE to ensure it generates accurate summaries.

### **Conclusion:**

By completing this challenge, you've learned to fine-tune a summarization model using FLAN-t5 and LoRA, an essential component of a comprehensive NLP solution. This ties together with sentiment analysis, translation, and Q&A to create a robust system capable of understanding, interpreting, and responding to human language in diverse applications.

## **Step-by-Step: Fine-Tuning the Summarization Model**

### **Introduction:**

In this exercise, we'll fine-tune the T5-small model for summarization using LoRA adapters. We'll break down each step and explain the code snippets in detail to ensure beginners understand how each part fits into the overall process.

### **Steps:**

Install required libraries:

* Install necessary libraries.

!pip install datasets transformers

*Explanation:*

* This command installs the datasets and transformers libraries, essential for handling datasets and pre-trained models.

Load data:

* Download and preprocess the dataset.

from datasets import load\_dataset

dataset = load\_dataset('cnn\_dailymail', '3.0.0')

small\_train\_dataset = dataset['train'].shuffle(seed=42).select(range(1000)) # Sample 1000 rows from the train set

small\_validation\_dataset = dataset['validation'].shuffle(seed=42).select(range(500)) # Sample 500 rows from the validation set

*Explanation:*

This code uses the load\_dataset function to download the CNN/Daily Mail dataset. The select method samples 1000 rows from the training set and 500 rows from the validation set, reducing the dataset size for faster processing.

Initialize model and tokenizer:

* Load the pre-trained T5-small model and tokenizer.

from transformers import TFAutoModelForSeq2SeqLM, AutoTokenizer

model\_name = "flan-t5-small"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

t5\_model = TFAutoModelForSeq2SeqLM.from\_pretrained(model\_name)

Explanation:

* model\_name specifies the pre-trained model we want to use, t5-small.
* AutoTokenizer preprocesses the text data.
* TFAutoModelForSeq2SeqLM loads the T5-small model for sequence-to-sequence learning tasks.

Tokenize data:

* Tokenize the dataset and create TensorFlow datasets for training and validation.

def tokenize\_function(examples):

inputs = tokenizer(examples['article'], max\_length=512, truncation=True, padding='max\_length', return\_tensors='tf')

targets = tokenizer(examples['highlights'], max\_length=128, truncation=True, padding='max\_length', return\_tensors='tf')

inputs['labels'] = targets['input\_ids']

inputs['decoder\_input\_ids'] = targets['input\_ids']

return inputs

train\_dataset = small\_train\_dataset.map(tokenize\_function, batched=True, remove\_columns=["id"])

val\_dataset = small\_validation\_dataset.map(tokenize\_function, batched=True, remove\_columns=["id"])

def convert\_to\_tf\_dataset(dataset):

input\_ids = tf.convert\_to\_tensor(dataset["input\_ids"], dtype=tf.int32)

attention\_mask = tf.convert\_to\_tensor(dataset["attention\_mask"], dtype=tf.int32)

decoder\_input\_ids = tf.convert\_to\_tensor(dataset["decoder\_input\_ids"], dtype=tf.int32)

labels = tf.convert\_to\_tensor(dataset["labels"], dtype=tf.int32)

return tf.data.Dataset.from\_tensor\_slices(({"input\_ids": input\_ids, "attention\_mask": attention\_mask, "decoder\_input\_ids": decoder\_input\_ids}, labels)).batch(32)

train\_data = convert\_to\_tf\_dataset(train\_dataset)

val\_data = convert\_to\_tf\_dataset(val\_dataset)

*Explanation:*

* tokenize\_function tokenizes the articles and summaries, truncating and padding them.
* convert\_to\_tf\_dataset converts the tokenized datasets into TensorFlow datasets, batching them for training and validation.

Integrate LoRA adapters:

* Define and add LoRA adapters to the model’s layers.

import keras

from keras.layers import Dense

from keras.models import Sequential

from keras.layers import Input

class LoraLayer(keras.layers.Layer):

def \_\_init\_\_(self, original\_layer, rank=8, num\_heads=1, dim=1, trainable=False, \*\*kwargs):

original\_layer\_config = original\_layer.get\_config()

name = original\_layer\_config["name"]

kwargs.pop("name", None)

super().\_\_init\_\_(name=name, trainable=trainable, \*\*kwargs)

self.rank = rank

self.original\_layer = original\_layer

self.original\_layer.trainable = False

self.A = keras.layers.Dense(units=rank, use\_bias=False, trainable=trainable, name=f"lora\_A")

self.B = keras.layers.Dense(units=dim, use\_bias=False, trainable=trainable, name=f"lora\_B")

def call(self, inputs):

original\_output = self.original\_layer(inputs)

if self.trainable:

lora\_output = self.B(self.A(inputs))

return original\_output + lora\_output

return original\_output

import transformers

from tf\_keras.src.layers.core.dense import Dense as NDense

def replace\_dense\_with\_lora(layer, rank=8):

if isinstance(layer, NDense):

return LoraLayer(original\_layer=layer, rank=rank)

return layer

def modify\_t5\_layers(t5, rank=8):

for sub\_layer in t5.encoder.submodules:

if isinstance(sub\_layer, transformers.models.t5.modeling\_tf\_t5.TFT5Attention):

sub\_layer.k = replace\_dense\_with\_lora(sub\_layer.k, rank)

sub\_layer.v = replace\_dense\_with\_lora(sub\_layer.v, rank)

sub\_layer.q = replace\_dense\_with\_lora(sub\_layer.q, rank)

sub\_layer.o = replace\_dense\_with\_lora(sub\_layer.o, rank)

for sub\_layer in t5.decoder.submodules:

if isinstance(sub\_layer, transformers.models.t5.modeling\_tf\_t5.TFT5Attention):

sub\_layer.k = replace\_dense\_with\_lora(sub\_layer.k, rank)

sub\_layer.v = replace\_dense\_with\_lora(sub\_layer.v, rank)

sub\_layer.q = replace\_dense\_with\_lora(sub\_layer.q, rank)

sub\_layer.o = replace\_dense\_with\_lora(sub\_layer.o, rank)

return t5

modified\_t5\_model = modify\_t5\_layers(t5\_model, rank=4)

for layer in modified\_t5\_model.\_flatten\_layers():

if (layer.\_\_class\_\_.\_\_module\_\_.startswith('tf\_keras') or layer.\_\_class\_\_.\_\_module\_\_.startswith('keras')) and not layer.name.startswith("lora"):

layer.trainable = False

elif layer.name.startswith("lora"):

layer.trainable = True

elif layer.name == 'shared':

layer.trainable = False

*Explanation:*

* LoraLayer defines a custom Keras layer for the LoRA adapters, using low-rank matrices A and B for efficient fine-tuning.
* replace\_dense\_with\_lora function replaces dense layers with LoRA layers.
* modify\_t5\_layers modifies the T5 model's layers to include LoRA adapters.
* The for loop sets the training configuration for each layer in the modified model.

Prepare for training:

* Compile and train the model.

import gc

from tf\_keras import backend as K

def clear\_gpu\_memory():

gc.collect()

tf.keras.backend.clear\_session()

K.clear\_session()

tf.compat.v1.reset\_default\_graph()

clear\_gpu\_memory()

from tf\_keras import mixed\_precision

mixed\_precision.set\_global\_policy('mixed\_float16')

tf.config.run\_functions\_eagerly(True)

modified\_t5\_model.compile(optimizer='adam', loss=t5\_model.hf\_compute\_loss)

modified\_t5\_model.fit(train\_data, epochs=1, validation\_data=val\_data)

*Explanation:*

* clear\_gpu\_memory clears GPU memory to prevent memory issues.
* mixed\_precision.set\_global\_policy('mixed\_float16') enables mixed precision training for performance.
* modified\_t5\_model.compile compiles the model with the Adam optimizer and a custom loss function.
* modified\_t5\_model.fit trains the model on the training dataset and evaluates it on the validation dataset.

Evaluate performance:

* Assess the model's performance.

!pip install rouge\_score

from datasets import load\_metric

rouge = load\_metric("rouge", trust\_remote\_code=True)

num\_samples = 1000

def generate\_summaries(dataset, num\_samples):

inputs = tokenizer([ex['article'] for ex in dataset.take(num\_samples)], return\_tensors='tf', padding=True, truncation=True, max\_length=512)

summaries = modified\_t5\_model.generate(inputs['input\_ids'], attention\_mask=inputs['attention\_mask'], max\_length=128, num\_beams=5)

decoded\_summaries = [tokenizer.decode(g, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=False) for g in summaries]

return decoded\_summaries

val\_summaries = [ex['highlights'] for ex in val\_dataset.take(num\_samples)]

generated\_summaries = generate\_summaries(val\_dataset, num\_samples=num\_samples)

results = rouge.compute(predictions=generated\_summaries, references=val\_summaries)

print(results)

*Explanation:*

* rouge loads the ROUGE evaluation metric.
* generate\_summaries generates summaries for the validation dataset.
* results calculates and prints the ROUGE score for the generated summaries compared to the reference summaries.

### **Conclusion:**

By following these steps, you've successfully fine-tuned the T5-small model for summarization using LoRA adapters. This approach leverages pre-trained knowledge and efficient fine-tuning techniques, allowing you to create robust NLP solutions that integrate sentiment analysis, translation, and Q&A capabilities.

# Summary

## Course recap and key takeaways

* Understanding Fine-Tuning: The importance and techniques of fine-tuning LLMs to enhance their performance for specific tasks.
* Model Architectures: Familiarity with different model architectures (encoder-decoder, encoder only, decoder only) and their use cases.
* Practical Applications: Real-world applications of fine-tuning LLMs in sectors like healthcare, finance, and customer service.
* LoRA Adapters: Using LoRA for parameter-efficient fine-tuning, making it feasible to adapt large models with limited computational resources.
* Hands-on Practice: Gaining practical experience through exercises on sentiment analysis, Q&A, and summarization.

## Advanced topics and future trends in LLMs

* Few-shot and Zero-shot Learning: Models can perform tasks with minimal to no task-specific training, reducing the need for large labeled datasets.
* Federated Learning: Training models across decentralized devices or servers to enhance privacy and reduce data centralization, especially relevant for sensitive applications like healthcare.
* Parameter-efficient Fine-tuning: Using techniques like LoRA adapters to adapt large models with limited computational resources.
* Multimodal AI: Integrating text, image, and audio inputs to create more immersive and intuitive user experiences.
* Efficient Training Techniques: Advancements like model pruning and quantization to deploy powerful models on edge devices, reducing computational costs and enabling real-time applications.

-------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------------------------------------

# Other notes

Machine Learning Algorithms 👇

🔍 Algorithms → Use Case → Formula → ⚠️ Avoid When

1. Linear Regression → Predict continuous values → Y = b0 + b1X + ... → Non-linear data

2. Logistic Regression → Binary classification → Sigmoid curve → Non-linear boundaries

3. Decision Tree → Interpret models → Split rules → Overfitting risk

4. Random Forest → High accuracy → Multiple trees voting → Slower, less transparent

5. Gradient Boosting → Precision tasks → Trees + loss minimization → Needs tuning

6. SVM → Margin-based classification → Kernel tricks → Too slow on big data

7. KNN → Small-scale prediction → Distance voting → Noisy/large data

8. Naive Bayes → Text classification → Probabilistic → Feature correlation breaks it

9. K-Means → Customer segmentation → Cluster center → Wrong with irregular shapes

10. PCA → Reduce features → Max variance → When interpretability is key

11. Neural Nets → Pattern recognition → Weights + activations → Low data? Not ideal

12. CNN → Image/video tasks → Convolutions → Not for sequences

13. RNN → Sequence prediction → Feedback loops → Long-term memory fades

14. Transformers (GPT/BERT) → NLP/AI chat → Attention mechanism → Heavy compute

15. DBSCAN → Shape-flexible clustering → Density → Sparse high-dim data

COurse: https://www.coursera.org/specializations/advanced-machine-learning-tensorflow-gcp?irclickid=SGDxsi3EbxycUpeyga0WLSmCUkp2Llx5lXBiw00&irgwc=1&utm\_medium=partners&utm\_source=impact&utm\_campaign=1999349&utm\_content=b2c&utm\_campaignid=Shailesh%20Kumar&utm\_term=14726\_CR\_1164545\_