Pretraining

a. Using distilgpt for textgeneration

b. Chat generation using mistralai/Mistral-7B-Instruct-v0.2

c. Summarize text using T5-small

```
# ========================================
# a.Using distilgpt2 for Text Generation
# ========================================
!pip install -q transformers datasets evaluate accelerate sentencepiece
!pip install -q torch

import torch
from transformers import pipeline, AutoModelForCausalLM, AutoTokenizer
print("✅ Environment ready. Torch version:", torch.__version__)
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

model_name = "distilgpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

prompt = "Artificial intelligence is transforming the world because"
inputs = tokenizer(prompt, return_tensors="pt")

outputs = model.generate(
    **inputs,
    max_length=60,
    num_return_sequences=3,
    temperature=0.8,
    top_k=50,
    top_p=0.95,
    do_sample=True
)

```

```
30
31
```

Show hidden output

```
for i, output in enumerate(outputs):
    print(f"\n--- Generated Text {i+1} ---\n")
    print(tokenizer.decode(output, skip_special_tokens=True))
```

```
--- Generated Text 1 ---

Artificial intelligence is transforming the world because the technology is moving us closer to the natural world, not closer to the natural world

--- Generated Text 2 ---

Artificial intelligence is transforming the world because of advances in the development of artificial intelligence, but it's still not new. A new



--- Generated Text 3 ---

Artificial intelligence is transforming the world because it's the kind of place where humans can communicate with each other.
```

```
# ====================================================
# b. Chat Generation using mistralai/Mistral-7B-Instruct-v0.2
# ====================================================

from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline

model_id = "mistralai/Mistral-7B-Instruct-v0.2"
pipe = pipeline("text-generation", model=model_id, torch_dtype=torch.bfloat16, device_map="auto")

prompt = "You are a helpful assistant. User: What is quantum computing? Assistant:"

result = pipe(prompt, max_new_tokens=120, temperature=0.7)


```

Show hidden output

```
print(result[0]['generated_text'])
```

```
You are a helpful assistant. User: What is quantum computing? Assistant: Quantum computing is a type of computing technology that utilizes quantum
```

```
# ====================================
# c. Summarize text using T5-small
# ====================================

from transformers import pipeline

summarizer = pipeline("summarization", model="t5-small")

text = """
Artificial Intelligence (AI) is a field of computer science that aims to create machines
that can perform tasks that would normally require human intelligence. This includes
learning, reasoning, problem-solving, perception, and language understanding.
"""

summary = summarizer(text, max_length=50, min_length=20, do_sample=False)

```

Show hidden output

```
print(" Summary:", summary[0]['summary_text'])

```

```
Summary: artificial intelligence is a field of computer science that aims to create machines that can perform tasks that would normally require h
```

Fine-tunning

a. Using BERT(BERT-base) for sentiment analysis using IMDb sentiment dataset

i. Using bert-base-uncased

ii. Using Hugging Face Trainer API

```
# Install required libraries
!pip install -q transformers datasets accelerate evaluate

# Import necessary modules
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
```

```
import numpy as np
import evaluate

# Load the IMDb dataset
dataset = load_dataset("imdb")

# Load the BERT tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Tokenization function
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True, max_length=512)

# Tokenize datasets
tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Use small subsets for fast training
small_train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(1000))
small_eval_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(200))

# Load model
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)

# Training arguments (fixed: use 'eval_strategy' instead of 'evaluation_strategy')
training_args = TrainingArguments(
    output_dir="bert_imdb_trainer",
    eval_strategy="epoch",           # Fixed: was 'evaluation_strategy'
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=1,
    weight_decay=0.01,
    logging_dir="logs",
    logging_steps=10,
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
)
```

```
44
45 # Load accuracy metric
46 metric = evaluate.load("accuracy")
47
48 # Compute metrics
49 def compute_metrics(eval_pred):
50     logits, labels = eval_pred
51     predictions = np.argmax(logits, axis=-1)
52     return metric.compute(predictions=predictions, references=labels)
53
54 # Initialize Trainer
55 trainer = Trainer(
56     model=model,
57     args=training_args,
58     train_dataset=small_train_dataset,
59     eval_dataset=small_eval_dataset,
60     compute_metrics=compute_metrics,
61 )
62
63 # Train
```

Show hidden output

```
1 import torch
2
3 # Sample inference with proper device handling
4 text = "This is a fantastic movie with great acting!"
5 inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
6
7 # Move inputs to the same device as the model
8 device = model.device  # Automatically detects GPU if available
9 inputs = {k: v.to(device) for k, v in inputs.items()}
10
11 # Forward pass
12 with torch.no_grad():
13     outputs = model(**inputs)
14
```

```
# Get prediction
logits = outputs.logits
prediction = torch.argmax(logits, dim=-1).item()

print("Text:", text)
print("Sentiment:", "Positive" if prediction == 1 else "Negative")
```

```
Text: This is a fantastic movie with great acting!
Sentiment: Positive
```

b. Fine-tuning GPT for summarization.

i. Using gpt2 for text summarization using DailyMail/XSum dataset

ii. Using Hugging face Pipeline

```
!pip install -U transformers datasets accelerate sentencepiece -q

import os
os.environ["WANDB_DISABLED"] = "true"

from datasets import load_dataset
from transformers import AutoTokenizer, GPT2LMHeadModel
from transformers import DataCollatorForLanguageModeling, TrainingArguments, Trainer
from transformers import pipeline

dataset = load_dataset("cnn_dailymail", "3.0.0")["train"].select(range(500))

tokenizer = AutoTokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token
model = GPT2LMHeadModel.from_pretrained("gpt2")

def preprocess(b):
    t = ["summarize: " + x for x in b["article"]]
    return tokenizer(t, truncation=True, padding="max_length", max_length=256)

tokenized = dataset.map(preprocess, batched=True, remove_columns=["article","highlights"])
collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
```

```

args = TrainingArguments(
    output_dir="./gpt2-sum",
    per_device_train_batch_size=2,
    gradient_accumulation_steps=4,
    num_train_epochs=1,
    learning_rate=5e-5,
    logging_steps=50,
    fp16=True
)

trainer = Trainer(model=model, args=args, train_dataset=tokenized, data_collator=collator)
trainer.train()

```

Show hidden output

```
summarizer = pipeline("text-generation", model=model, tokenizer=tokenizer, max_new_tokens=60)
print(summarizer("summarize: The Eiffel Tower is visited by millions every year.")[0]["generated_text"])
```

```
Device set to use cuda:0
summarize: The Eiffel Tower is visited by millions every year. But for the second year running, the hotel's owners are making it back to normal. T
```

```
# Install required libraries
!pip install -q transformers datasets accelerate evaluate

# Import modules
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
import numpy as np
import evaluate
import torch

# Load IMDb dataset
dataset = load_dataset("imdb")

```

```
# Load DistilBERT tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

# Tokenization function
def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True, padding="max_length", max_length=512)

# Tokenize datasets
tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Small subsets for quick training
small_train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(1000))
small_eval_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(200))

# Load DistilBERT model for classification
model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=2)

# Training arguments
training_args = TrainingArguments(
    output_dir="distilbert_imdb",
    eval_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=16,   # DistilBERT is lighter → can use larger batch
    per_device_eval_batch_size=16,
    num_train_epochs=1,
    weight_decay=0.01,
    logging_dir="logs",
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    report_to="none",  # Disable wandb
)

# Accuracy metric
metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
```

```
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,
)

# Train
trainer.train()

# Move model to correct device for inference
```

Show hidden output

```
text = "This is a fantastic movie with great acting!"
inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=512)
inputs = {k: v.to(model.device) for k, v in inputs.items()}

with torch.no_grad():
    outputs = model(**inputs)
    pred = torch.argmax(outputs.logits, dim=-1).item()

print("Text:", text)
print("Sentiment:", "Positive" if pred == 1 else "Negative")
```

```
Text: This is a fantastic movie with great acting!
Sentiment: Positive
```

For text summarization Swap the models (bart-large-cnn, t5-small, pegasus-xsum) and change parameters and use trainer API

```
# Install required libraries
!pip install -q transformers datasets evaluate rouge_score accelerate torch

import torch
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, Seq2SeqTrainingArguments, Seq2SeqTrainer, Dat
import numpy as np
import evaluate

# === CONFIG: 3 MODELS ===
models_to_train = [
    {"name": "facebook/bart-large-cnn", "batch_size": 4,  "epochs": 1, "prefix": ""},
    {"name": "t5-small",                "batch_size": 8,  "epochs": 1, "prefix": "summarize: "},
    {"name": "google/pegasus-xsum",     "batch_size": 4,  "epochs": 1, "prefix": ""}
]

# Load CNN/DailyMail dataset
dataset = load_dataset("cnn_dailymail", "3.0.0")
train_dataset = dataset["train"].shuffle(seed=42).select(range(1000))
eval_dataset = dataset["validation"].shuffle(seed=42).select(range(200))

# Load ROUGE
rouge = evaluate.load("rouge")

def compute_metrics(eval_pred):
    preds, labels = eval_pred
    decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)
    result = rouge.compute(predictions=decoded_preds, references=decoded_labels, use_stemmer=True)
    return {k: round(v, 4) for k, v in result.items()}

# === TRAINING LOOP ===
for idx, config in enumerate(models_to_train):
    print(f"\n{'='*70}")
    print(f"TRAINING MODEL {idx+1}/3: {config['name']}")
    print(f"{'='*70}\n")
```

```
38
39  MODEL_NAME = config["name"]
40  BATCH_SIZE = config["batch_size"]
41  EPOCHS = config["epochs"]
42  PREFIX = config["prefix"]
43
44  # Load tokenizer & model
45  tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
46  model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)
47
48  # Preprocess function
49  def preprocess(examples):
50      inputs = [PREFIX + doc for doc in examples["article"]]
51      model_inputs = tokenizer(inputs, max_length=512, truncation=True, padding="max_length")
52      with tokenizer.as_target_tokenizer():
53          labels = tokenizer(examples["highlights"], max_length=128, truncation=True, padding="max_length"
54      model_inputs["labels"] = labels["input_ids"]
55      return model_inputs
56
57  # Tokenize
58  tokenized_train = train_dataset.map(preprocess, batched=True, remove_columns=["article", "highlights", "
59  tokenized_eval = eval_dataset.map(preprocess, batched=True, remove_columns=["article", "highlights", "id
60
61  data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
62
63  # Training args
64  training_args = Seq2SeqTrainingArguments(
65      output_dir=f"./cnn_model_{idx}_{MODEL_NAME.split('/')[-1]}",
66      eval_strategy="epoch",
67      save_strategy="epoch",
68      per_device_train_batch_size=BATCH_SIZE,
69      per_device_eval_batch_size=BATCH_SIZE,
70      num_train_epochs=EPOCHS,
71      weight_decay=0.01,
72      predict_with_generate=True,
73      fp16=torch.cuda.is_available(),
74      logging_steps=10,
```

```
        load_best_model_at_end=True,
        metric_for_best_model="rouge1",
        report_to="none",
        disable_tqdm=True,
    )

    # Trainer
    trainer = Seq2SeqTrainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_train,
        eval_dataset=tokenized_eval,
        tokenizer=tokenizer,
        data_collator=data_collator,
        compute_metrics=compute_metrics,
    )

    # Train
    trainer.train()

    # Save
    save_dir = f"./trained_cnn_{MODEL_NAME.split('/')[-1]}"
    model.save_pretrained(save_dir)
    tokenizer.save_pretrained(save_dir)
    print(f"Model saved to: {save_dir}\n")
```

Show hidden output

```
from transformers import pipeline
import torch, os, warnings, textwrap

# Hide all warnings
warnings.filterwarnings("ignore")
os.environ["TOKENIZERS_PARALLELISM"] = "false"

# Test article
text = """London (CNN) -- A British man has been charged with murder after his wife and daughter were found
```

```
The bodies of 43-year-old Samantha Baldwin and her 7-year-old daughter, Lily, were discovered Monday at thei
The suspect, 45-year-old David Baldwin, was arrested at the scene and remains in custody, police said. He is
A post-mortem examination is being carried out to establish the cause of death, police said. Neighbors descr

# Your actual saved model folders
models = [
    ("BART-Large-CNN",   "/content/trained_cnn_bart-large-cnn"),
    ("T5-Small",         "/content/trained_cnn_t5-small"),
    ("Pegasus-XSum",     "/content/trained_cnn_pegasus-xsum")
]

print("=" * 90)
print("CNN/DAILYMAIL SUMMARIZATION RESULTS".center(90))
print("=" * 90 + "\n")

for name, folder in models:
    if not os.path.exists(folder):
        print(f"ERROR: Folder not found → {folder}")
        print("   Make sure the model was saved correctly.\n")
        continue

    summarizer = pipeline(
        "summarization",
        model=folder,
        tokenizer=folder,
        device=0 if torch.cuda.is_available() else -1,
        max_length=80,
        min_length=30,
        do_sample=False,
        truncation=True,
        clean_up_tokenization_spaces=True
    )

    summary = summarizer(text)[0]["summary_text"]

    print(f"{name.upper()}")
    print("—" * 70)
```

```
    print(textwrap.fill(summary, width=88))
```

```
================================================================================
                        CNN/DAILYMAIL SUMMARIZATION RESULTS
================================================================================

Device set to use cuda:0
Asking to truncate to max_length but no maximum length is provided and the model has no predefined maximum length. Default to no truncation.
BART-LARGE-CNN
--------------------------------------------------------------------------------
Samantha Baldwin, 43, and her 7-year-old daughter, Lily, were found dead at their home.
David Baldwin, 45, was arrested at the scene and remains in custody, police say. A post-
mortem examination is being carried out to establish the cause of death.

Device set to use cuda:0
Your max_length is set to 200, but your input_length is only 148. Since this is a summarization task, where outputs shorter than the input are typ
T5-SMALL
--------------------------------------------------------------------------------
the bodies of Samantha Baldwin, 43, and her 7-year-old daughter, Lily, were found Monday
at their home in southern England. David Baldwin, 45, was arrested at the scene and
remains in custody, police say. Neighbors described the family as "quiet and private"

Device set to use cuda:0
PEGASUS-XSUM
--------------------------------------------------------------------------------
The bodies of 43-year-old Samantha Baldwin and her 7-year-old daughter, Lily, were
discovered Monday at their home in Basingstoke, Hampshire, police said.
```

For Question Answering Use RoBERTa variants, GPT-3/4 fine-tuned on SQuAD dataset

```
!pip install -q transformers datasets accelerate sentencepiece

from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
from transformers import TrainingArguments, Trainer, DefaultDataCollator
import torch
import os

os.environ["WANDB_DISABLED"] = "true"

dataset = load_dataset("squad")
```

```
12
13 tokenizer = AutoTokenizer.from_pretrained("roberta-base")
14 model = AutoModelForQuestionAnswering.from_pretrained("roberta-base")
15
16 def preprocess(example):
17     inputs = tokenizer(
18         example["question"],
19         example["context"],
20         max_length=384,
21         truncation=True,
22         padding="max_length",
23         return_offsets_mapping=True
24     )
25     offset = inputs["offset_mapping"]
26     answers = example["answers"]["text"][0]
27     start_char = example["answers"]["answer_start"][0]
28     end_char = start_char + len(answers)
29
30     start_pos = 0
31     end_pos = 0
32     for i, (s, e) in enumerate(offset):
33         if s <= start_char < e:
34             start_pos = i
35         if s < end_char <= e:
36             end_pos = i
37
38     inputs["start_positions"] = start_pos
39     inputs["end_positions"] = end_pos
40     inputs.pop("offset_mapping")
41     return inputs
42
43 tokenized = dataset.map(preprocess, batched=False)
44
45 collator = DefaultDataCollator()
46
47 args = TrainingArguments(
48     output_dir="./roberta-qa",
```

```
    per_device_train_batch_size=8,
    num_train_epochs=1,
    logging_steps=100,
    report_to=[]
)

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=tokenized["train"],
    eval_dataset=tokenized["validation"],
    data_collator=collator,
    tokenizer=tokenizer
)

trainer.train()

from transformers import pipeline
qa = pipeline("question-answering", model=model, tokenizer=tokenizer)
```

Show hidden output

```
print(qa({"question":"Who founded Microsoft?", "context":"Microsoft was founded by Bill Gates and Paul Allen.
```

```
{'score': 0.8770565986633301, 'start': 25, 'end': 50, 'answer': 'Bill Gates and Paul Allen'}
```

Build a simple rule-based chatbot for FAQs.

```
def chatbot(message):
    msg = message.lower()

    # Greetings
    if "hello" in msg or "hi" in msg:
        return "Hello! How can I help you today?"

    # About company
```

```
    elif "what is your name" in msg:
        return "I'm a simple FAQ chatbot."

    elif "what do you do" in msg:
        return "I answer frequently asked questions."

    # Timings
    elif "working hours" in msg or "open" in msg:
        return "We are open from 9 AM to 6 PM, Monday to Saturday."

    # Pricing
    elif "price" in msg or "cost" in msg:
        return "Pricing depends on the service. Basic plans start at $10/month."

    # Support
    elif "support" in msg or "help" in msg:
        return "You can contact support at support@example.com."

    # Location
    elif "location" in msg or "where are you" in msg:
        return "We are located in Chennai, India."

    # Goodbye
    elif "bye" in msg or "goodbye" in msg:
        return "Goodbye! Have a great day!"

    # Default fallback
    else:
        return "I'm not sure about that. Could you ask something else?"


# Chat loop
print("Type 'exit' to stop.\n")
while True:
    user = input("You: ")
    if user.lower() == "exit":
        print("Chatbot: Goodbye!")
```

```
        break
    print("Chatbot:", chatbot(user))
```

```
Type 'exit' to stop.

You: support
Chatbot: You can contact support at support@example.com.
You: bye
Chatbot: Goodbye! Have a great day!
You: exit
Chatbot: Goodbye!
```

Apply various prompting techniques and view the difference

```
!pip install transformers accelerate sentencepiece -q

from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, pipeline

model = "google/flan-t5-large"  # fits on Colab T4 with 16GB
pipe = pipeline("text2text-generation", model=model, tokenizer=model, max_new_tokens=150)

def run(prompt):
    print("\n--- PROMPT ---")
    print(prompt)
    print("--- OUTPUT ---")
    print(pipe(prompt)[0]["generated_text"])
    print("\n")
```

Show hidden output

```


# 1. Zero-shot
run("Explain blockchain in simple terms.")

# 2. One-shot
run("""
Q: What is AI?
```

```
A: AI means machines that can think and learn like humans.

Q: What is blockchain?
A:
""")

# 3. Few-shot
run("""
Q: What is AI?
A: AI refers to machines performing tasks that require human intelligence.

Q: What is Machine Learning?
A: ML is a subset of AI that learns patterns from data.

Q: What is blockchain?
A:
""")

# 4. Chain-of-thought
run("Why do objects fall to the ground? Let's think step by step.")

# 5. Self-consistency style
run("""
Why does ice float on water?

Let's think step by step:
""")

# 6. Instruction prompting
run("""
Summarize the following in 3 lines:
Artificial Intelligence enables machines to learn from data and make decisions.
""")

# 7. Role prompting
run("""
You are a polite science teacher.
```

```
Explain how the Internet works.
""")

# 8. Delimiter prompting
run("""
Summarize this text: <text>Machines learn patterns from data in ML.</text>
""")

# 9. Contrastive prompt
run("""
Bad summary: It's about machines.
Good summary: Give a precise, clear summary.

Text: Deep learning uses neural networks with layers to learn patterns from data.

Summary:
""")
```

```
--- PROMPT ---

You are a polite science teacher.
Explain how the Internet works.

--- OUTPUT ---
The Internet is a network of computers that communicate with each other. The computers are called nodes. The nodes are called nodes. The nodes a



--- PROMPT ---

Summarize this text: <text>Machines learn patterns from data in ML.</text>

--- OUTPUT ---
Machine learning (ML) is a branch of computer science that focuses on learning patterns from data.



--- PROMPT ---

Bad summary: It's about machines.
Good summary: Give a precise, clear summary.

Text: Deep learning uses neural networks with layers to learn patterns from data.

Summary:

--- OUTPUT ---
Deep learning uses neural networks with layers to learn patterns from data.
```

Compare extractive and abstractive summarization using Hugging Face and TextRank

```
!pip install -U transformers datasets summa -q

from summa.summarizer import summarize
from transformers import pipeline

text = """Artificial Intelligence is transforming industries across the globe.
Companies are rapidly adopting AI solutions to automate tasks, increase efficiency,
enhance customer experience, and make better decisions. The rise of AI-driven
innovation has created new opportunities in healthcare, finance, education, and transportation."""

```

```
extractive = summarize(text, ratio=0.4)

abstractive_model = pipeline("summarization", model="t5-base", tokenizer="t5-base")
abstractive = abstractive_model("summarize: " + text, max_length=60, min_length=20)[0]["summary_text"]


```

Show hidden output

```
print("Extractive Summary (TextRank):\n", extractive, "\n")
print("Abstractive Summary (HF):\n", abstractive)

```

```
Extractive Summary (TextRank):
 Companies are rapidly adopting AI solutions to automate tasks, increase efficiency,
The rise of AI-driven

Abstractive Summary (HF):
 companies are rapidly adopting AI solutions to automate tasks . the rise of AI-driven innovation has created new opportunities .
```

Compare retrieval-based vs generative chatbot responses.

```
!pip install -q transformers scikit-learn

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

corpus = [
    "Hello, how can I help you?",
    "AI stands for Artificial Intelligence.",
    "Machine learning is a subset of AI.",
    "The weather is nice today.",
    "Chatbots can be retrieval-based or generative."
]

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
```

```

def retrieval_chatbot(query):
    q_vec = vectorizer.transform([query])
    sims = cosine_similarity(q_vec, X).flatten()
    return corpus[sims.argmax()]

tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-small")
model = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-small")

def generative_chatbot(query):
    inputs = tokenizer(query, return_tensors="pt")
    output = model.generate(**inputs, max_new_tokens=50)
    return tokenizer.decode(output[0], skip_special_tokens=True)

query = "What is AI?"

```

Show hidden output

```
print("User:", query)
print("\nRetrieval-based Response:\n", retrieval_chatbot(query))
print("\nGenerative Response:\n", generative_chatbot(query))
```

```
User: What is AI?

Retrieval-based Response:
 Machine learning is a subset of AI.

Generative Response:
 AI (disambiguation) AI is a term used to describe a person's behavior.
```