

ASSIGNMENT 6.1 - PRE-TRAINED GENERATIVE MODEL USAGE

Pre-trained generative models are powerful tools that can be accessed via APIs to perform tasks like text generation, summarization, translation, and more. Here are some popular pre-trained generative models and their respective APIs: Below are some of the available PRe-trained Generative models and APIs.

1. OpenAI GPT (Generative Pre-trained Transformer)

- **API:** [OpenAI API](#)
 - **Models:** GPT-3, GPT-4
 - **Use Cases:**
 - Text generation
 - Summarization
 - Question answering
 - Code generation
 - **Advantages:**
 - State-of-the-art performance on natural language tasks.
 - Easy integration via REST API.
-

2. Hugging Face Transformers

- **API:** [Hugging Face Inference API](#)
 - **Models:**
 - BART (e.g., `facebook/bart-large-cnn`) for summarization.
 - T5 (Text-to-Text Transfer Transformer) for text-to-text tasks.
 - GPT-2 and GPT-Neo for text generation.
 - **Use Cases:**
 - Summarization
 - Translation
 - Text classification
 - **Advantages:**
 - Open-source models with a wide variety of options.
 - Hosted inference API for easy access.
-

3. Google PaLM (Pathways Language Model)

- **API:** [Google Cloud Generative AI](#)
- **Models:** PaLM 2
- **Use Cases:**
 - Text generation and summarization.
 - Chatbots and conversational AI.
 - Code generation and debugging.
- **Advantages:**

- High performance on multilingual and multimodal tasks.
 - Seamless integration with Google Cloud services.
-

4. Cohere

- **API:** [Cohere API](#)
 - **Models:**
 - Generate for text generation tasks.
 - **Use Cases:**
 - Content creation
 - Semantic search
 - Text summarization
 - **Advantages:**
 - Focused on enterprise use cases with fine-tuning options.
-

5. Anthropic Claude

- **API:** [Anthropic Claude API](#)
 - **Models:** Claude (Claude Instant, Claude Pro)
 - **Use Cases:**
 - Conversational AI
 - Document summarization
 - Ethical AI applications
 - **Advantages:**
 - Designed with safety and ethical considerations in mind.
-

6. Microsoft Azure OpenAI Service

- **API:** [Azure OpenAI Service](#)
 - **Models:** GPT models (via OpenAI partnership)
 - **Use Cases:**
 - Enterprise-level text generation and summarization.
 - Integration with Microsoft services like Power BI and Azure Cognitive Search.
 - **Advantages:**
 - Scalable infrastructure for enterprise needs.
-

7. IBM Watson Natural Language Understanding

- **API:** [IBM Watson NLU](#)
- **Models:** Custom Watson NLP models
- **Use Cases:**
 - Sentiment analysis
 - Keyword extraction
 - Summarization

- **Advantages:**

- Strong focus on enterprise analytics and insights.

Advantages of Pre-trained Generative Models Accessed via APIs

1. Reduced Development Time

- Leverage existing, powerful models without the need for extensive training from scratch, saving significant time and resources.

2. Accessibility and Scalability

- APIs provide easy access to these models, abstracting away complex infrastructure and allowing for scalable usage as needed.

3. Cost-Effectiveness

- Avoid the high costs associated with training and hosting large models; pay-as-you-go pricing for API usage makes it more affordable.

4. State-of-the-Art Performance

- Benefit from models trained on massive datasets, often achieving superior performance compared to custom-trained models, especially for complex tasks.

5. Focus on Application Logic

- Concentrate on building your application's specific features rather than dealing with the intricacies of model training and deployment.

6. Continuous Improvement

- API providers often update their models, ensuring you have access to the latest advancements in generative AI.

7. Simplified Integration

- APIs offer a standardized and straightforward way to integrate powerful generative capabilities into your existing workflows and applications.

✓ **STEP BY STEP PROCEDURE TO SETUP ENVIRONMENT**

1. Uninstall unsupported versions of PyTorch, TorchVision, and Torchaudio from the Development environment

2. Installs PyTorch, TorchVision, and Torchaudio using the specified CUDA version (cu118) in this case

CUDA And PyTorch:

CUDA is NVIDIA's parallel computing platform that allows PyTorch to leverage GPU acceleration.

The installed PyTorch must match the installed CUDA version on the environment for GPU support to work correctly.

```
# Force reinstall PyTorch with correct CUDA version (Colab specific)
!pip uninstall torch torchvision torchaudio -y
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118 # Choose your CUDA
```

```
Found existing installation: torch 2.5.1+cu124
Uninstalling torch-2.5.1+cu124:
  Successfully uninstalled torch-2.5.1+cu124
Found existing installation: torchvision 0.20.1+cu124
Uninstalling torchvision-0.20.1+cu124:
  Successfully uninstalled torchvision-0.20.1+cu124
Found existing installation: torchaudio 2.5.1+cu124
Uninstalling torchaudio-2.5.1+cu124:
  Successfully uninstalled torchaudio-2.5.1+cu124
Looking in indexes: https://download.pytorch.org/whl/cu118
Collecting torch
  Downloading https://download.pytorch.org/whl/cu118/torch-2.6.0%2Bcu118-cp311-cp311-linux\_x86\_64.whl.met
Collecting torchvision
  Downloading https://download.pytorch.org/whl/cu118/torchvision-0.21.0%2Bcu118-cp311-cp311-linux\_x86\_64
Collecting torchaudio
  Downloading https://download.pytorch.org/whl/cu118/torchaudio-2.6.0%2Bcu118-cp311-cp311-linux\_x86\_64.wh
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch) (3.17.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch) (4.10.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch) (3.4.2)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.11/dist-packages (from torch) (3.1.5)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch) (2024.10.0)
Collecting nvidia-cuda-nvrtc-cu11==11.8.89 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_cuda\_nvrtc\_cu11-11.8.89-py3-none-manylinux1\_x
  23.2/23.2 MB 88.3 MB/s eta 0:00:00
Collecting nvidia-cuda-runtime-cu11==11.8.89 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_cuda\_runtime\_cu11-11.8.89-py3-none-manylinux1\_x
  875.6/875.6 kB 55.2 MB/s eta 0:00:00
Collecting nvidia-cuda-cupti-cu11==11.8.87 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_cuda\_cupti\_cu11-11.8.87-py3-none-manylinux1\_x
  13.1/13.1 MB 106.1 MB/s eta 0:00:00
Collecting nvidia-cudnn-cu11==9.1.0.70 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_cudnn\_cu11-9.1.0.70-py3-none-manylinux2014\_x8
  663.9/663.9 MB 2.8 MB/s eta 0:00:00
Collecting nvidia-cublas-cu11==11.11.3.6 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_cublas\_cu11-11.11.3.6-py3-none-manylinux1\_x86
  417.9/417.9 MB 3.7 MB/s eta 0:00:00
Collecting nvidia-cufft-cu11==10.9.0.58 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_cufft\_cu11-10.9.0.58-py3-none-manylinux1\_x86
  168.4/168.4 MB 6.8 MB/s eta 0:00:00
Collecting nvidia-curand-cu11==10.3.0.86 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_curand\_cu11-10.3.0.86-py3-none-manylinux1\_x86
  58.1/58.1 MB 13.8 MB/s eta 0:00:00
Collecting nvidia-cusolver-cu11==11.4.1.48 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_cusolver\_cu11-11.4.1.48-py3-none-manylinux1\_x
  128.2/128.2 MB 7.4 MB/s eta 0:00:00
Collecting nvidia-cuspars-cu11==11.7.5.86 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_cuspars\_cu11-11.7.5.86-py3-none-manylinux1\_x
  204.1/204.1 MB 6.1 MB/s eta 0:00:00
Collecting nvidia-nccl-cu11==2.21.5 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_nccl\_cu11-2.21.5-py3-none-manylinux2014\_x86\_6
  147.8/147.8 MB 7.7 MB/s eta 0:00:00
Collecting nvidia-nvtx-cu11==11.8.86 (from torch)
  Downloading https://download.pytorch.org/whl/cu118/nvidia\_nvtx\_cu11-11.8.86-py3-none-manylinux1\_x86\_64
  99.1/99.1 kB 9.6 MB/s eta 0:00:00
Collecting triton==3.2.0 (from torch)
  Downloading https://download.pytorch.org/whl/triton-3.2.0-cp311-cp311-manylinux\_2\_27\_x86\_64\_manylinux\_2
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from triton) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1) (1.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from triton) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/dist-packages (from triton) (10.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from Jinja2) (3.0.2)
  Downloading https://download.pytorch.org/whl/cu118/torch-2.6.0%2Bcu118-cp311-cp311-linux\_x86\_64.whl (848.7/848.7 MB 1.8 MB/s eta 0:00:00)
  Downloading https://download.pytorch.org/whl/triton-3.2.0-cp311-cp311-manylinux\_2\_27\_x86\_64\_manylinux\_2\_2
  166.7/166.7 MB 6.8 MB/s eta 0:00:00
  Downloading https://download.pytorch.org/whl/cu118/torchvision-0.21.0%2Bcu118-cp311-cp311-linux\_x86\_64.wh
  6.5/6.5 MB 101.5 MB/s eta 0:00:00
  Downloading https://download.pytorch.org/whl/cu118/torchaudio-2.6.0%2Bcu118-cp311-cp311-linux\_x86\_64.whl
```

```
3.3/3.3 MB 91.6 MB/s eta 0:00:00
Installing collected packages: triton, nvidia-nvtx-cu11, nvidia-nccl-cu11, nvidia-cusparse-cu11, nvidia-c
Attempting uninstall: triton
  Found existing installation: triton 3.1.0
  Uninstalling triton-3.1.0:
    Successfully uninstalled triton-3.1.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed
fastai 2.7.18 requires torch<2.6,>=1.10, but you have torch 2.6.0+cu118 which is incompatible.
Successfully installed nvidia-cublas-cu11-11.11.3.6 nvidia-cuda-cupti-cu11-11.8.87 nvidia-cuda-nvrtc-cu11
```

✓ An Example usage of Pre-Trained Model API - EleutherAI/gpt-neo-2.7B is a large language model

```
# Import the pipeline function, which provides a high-level, easy-to-use interface for various NLP tasks, includ
from transformers import pipeline

# Ensure the transformers library is installed
!pip install transformers

# Hugging Face Transformers pipeline for text generation
# EleutherAI/gpt-neo-2.7B is a large language model from the GPT-Neo family.
# It has 2.7 billion parameters. This line downloads the model if it's not already cached.
# Calls the pipeline with the prompt and other parameters
generator = pipeline('text-generation', model='EleutherAI/gpt-neo-2.7B')

# Prompt the model to generate a 2-line description of the Pythagorean theorem
prompt = "Describe the Pythagorean theorem in 2 lines."

# The generator function returns a list of dictionaries. Since we only asked for one sequence, we take the first
response = generator(prompt, max_length=50, num_return_sequences=1, truncation=True)[0]['generated_text']

#Print the Prompt respnse
print(f"Description of Pythagorean Theorem:\n{response}")
```

```

➡ Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.49.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.17
Requirement already satisfied: huggingface-hub<1.0,>=0.26.0 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (1
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transforme
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from trans
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.11/dist-packages (from transform
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from h
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from req
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->trans
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests-
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/setting)
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(

config.json: 100% 1.46k/1.46k [00:00<00:00, 82.7kB/s]

model.safetensors: 100% 10.7G/10.7G [01:00<00:00, 202MB/s]

tokenizer_config.json: 100% 200/200 [00:00<00:00, 13.9kB/s]

vocab.json: 100% 798k/798k [00:00<00:00, 1.28MB/s]

merges.txt: 100% 456k/456k [00:00<00:00, 5.80MB/s]

special_tokens_map.json: 100% 90.0/90.0 [00:00<00:00, 6.61kB/s]
Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Description of Pythagorean Theorem:
Describe the Pythagorean theorem in 2 lines.

A Pythagorean Square is a square with equal sides.

1
2
3
12
1x1
x2-x1

```

✓ An Example usage of Pre-Trained Model API - GPT2:

Prompt to the AI model: What is the synonym of Euphoric?

```

from transformers import pipeline
from langchain.llms import HuggingFacePipeline
import os

```

```
os.environ["HUGGINGFACEHUB_API_TOKEN"] = "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"


# Initialize the pipeline for text generation
generator = pipeline('text-generation', model='gpt2') # Or a smaller model if needed

# Create a LangChain wrapper around the Hugging Face pipeline
llm = HuggingFacePipeline(pipeline=generator)

# Prompt the model to generate a description of the Pythagorean theorem
prompt = "What is synonym of Euphoric?"

response = llm(prompt) # Use the LangChain LLM to get the response

print(response)
```

 Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
What is synonym of Euphoric? Is it possible? And yet when we talk of Euphoric we mean two different things:

The Euphoric [is a Greek Greek adjective meaning "that of the sun"] – where the sun is actually

✓ Install LangChain Community

langchain_community is a collection of community-contributed integrations for the LangChain library. LangChain is a framework for developing applications powered by language models. The langchain_community package often provides integrations with specific LLMs, vector databases, or other tools that are not included in the core langchain package.

```
!pip install langchain_community
```

 Requirement already satisfied: tenacity!=8.4.0,<10,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (tr
Collecting dataclasses-json<0.7,>=0.5.7 (from langchain_community)
 Downloading dataclasses_json-0.6.7-py3-none-any.whl.metadata (25 kB)
Collecting pydantic-settings<3.0.0,>=2.4.0 (from langchain_community)
 Downloading pydantic_settings-2.7.1-py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: langsmith<0.4,>=0.1.125 in /usr/local/lib/python3.11/dist-packages (from l
Collecting httpx-sse<1.0.0,>=0.4.0 (from langchain_community)
 Downloading httpx_sse-0.4.0-py3-none-any.whl.metadata (9.0 kB)
Requirement already satisfied: numpy<2,>=1.26.4 in /usr/local/lib/python3.11/dist-packages (from langchai
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from a
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp<
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp<4.0
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aioht
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp<
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp
Collecting marshmallow<4.0.0,>=3.18.0 (from dataclasses-json<0.7,>=0.5.7->langchain_community)
 Downloading marshmallow-3.26.1-py3-none-any.whl.metadata (7.3 kB)
Collecting typing-inspect<1,>=0.4.0 (from dataclasses-json<0.7,>=0.5.7->langchain_community)
 Downloading typing_inspect-0.9.0-py3-none-any.whl.metadata (1.5 kB)
Requirement already satisfied: langchain-text-splitters<1.0.0,>=0.3.6 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: pydantic<3.0.0,>=2.7.4 in /usr/local/lib/python3.11/dist-packages (from la
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.11/dist-packages (from lang
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.11/dist-packages (from langc


```

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from reques
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from reques
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.11/dist-packages (from SQLAlche
Requirement already satisfied: anyio in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->1
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0
Requirement already satisfied: h11<0.15,>=0.13 in /usr/local/lib/python3.11/dist-packages (from httpcore=
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.11/dist-packages (from jsonpatc
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from py
Requirement already satisfied: pydantic-core==2.27.2 in /usr/local/lib/python3.11/dist-packages (from pyd
Collecting mypy_extensions>=0.3.0 (from typing-inspect<1,>=0.4.0->dataclasses-json<0.7,>=0.5.7->langchain
  Downloading mypy_extensions-1.0.0-py3-none-any.whl.metadata (1.1 kB)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio->httpx
  Downloading langchain_community-0.3.17-py3-none-any.whl (2.5 MB)
     2.5/2.5 MB 87.0 MB/s eta 0:00:00
  Downloading dataclasses_json-0.6.7-py3-none-any.whl (28 kB)
  Downloading httpx_sse-0.4.0-py3-none-any.whl (7.8 kB)
  Downloading pydantic_settings-2.7.1-py3-none-any.whl (29 kB)
  Downloading marshmallow-3.26.1-py3-none-any.whl (50 kB)
     50.9/50.9 kB 5.1 MB/s eta 0:00:00
  Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
  Downloading typing_inspect-0.9.0-py3-none-any.whl (8.8 kB)
  Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Installing collected packages: python-dotenv, mypy_extensions, marshmallow, httpx-sse, typing-inspect, py
Successfully installed dataclasses-json-0.6.7 httpx-sse-0.4.0 langchain_community-0.3.17 marshmallow-3.26

```

SET UP TEXT GENERATION PIPELINE USING facebook/blenderbot-1B-distill MODEL

This code sets up a text generation pipeline using the facebook/blenderbot-1B-distill model and then wraps it with LangChain's HuggingFacePipeline for easier integration with LangChain workflows.

```

# Import the LangChain class to wrap Hugging Face pipelines.
from langchain.llms import HuggingFacePipeline

#Imports the PyTorch library (used by the model)
import torch

#Import necessary classes from the Hugging Face transformers library:
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline, AutoModelForSeq2SeqLM

model_id = 'facebook/blenderbot-1B-distill'
tokenizer = AutoTokenizer.from_pretrained(model_id) # For loading the correct tokenizer for the model.

# For loading sequence-to-sequence language models (used here). It loads the pretrained model weights
model = AutoModelForSeq2SeqLM.from_pretrained(model_id)

#Creates the text2text-generation pipeline
pipe = pipeline(
    "text2text-generation", #Specifies the task as text-to-text generation (suitable for models like BlenderBot)
    model=model, # passes the loaded model to pipeline
    tokenizer=tokenizer, #Passes the loaded tokenizer.
    max_length=100 #Sets the maximum length of the generated output to 100 tokens
)

# Creates a LangChain HuggingFacePipeline object (local_llm) by wrapping the Hugging Face pipe (the text generat
# This makes the model easily usable within LangChain chains and agents. local_llm is now a LangChain LLM object
local_llm = HuggingFacePipeline(pipeline=pipe)

```

🔄 Device set to use cuda:0

✓ USE LANGCHAIN FRAMEWORK FOR BUILDING LLM APPLICATIONS

Use LangChain, a framework for building applications with large language models (LLMs), to answer a question using a pre-existing language model represented by local_llm

```
# Import necessary tools from LangChain
from langchain import PromptTemplate, HuggingFaceHub, LLMChain #LLMChain - For creating a chain that combines a

# This defines a template for how the question should be formatted when sent to the LLM.
# It's like a pre-written message with a blank space{question}` where the actual question will go.
# The "Let's think step by step" part encourages the LLM to provide a more detailed reasoning in its answer.
template = """Question: {question}

Answer: Let's think step by step."""

# Create a PromptTemplate object from the template string. It specifies that the template has one input variable
prompt = PromptTemplate(template=template, input_variables=["question"])

# creates an LLMChain object, which connects the prompt template with the local_llm (your language model).
# local_llm would have been defined earlier in your code (not shown here) and is assumed to be a LangChain wrapper
llm_chain = LLMChain(prompt=prompt,
                    llm=local_llm
                    )

question = "what is Artificial Intelligence"

print(llm_chain.run(question))
```

🔄 I'm not sure what you mean by that, but I do know that artificial intelligence is a branch of computer science

✓ QUESTION TO LLM using API: "Which country is best for growing wine?"

```
question = "Which country is best for growing wine?"

print(llm_chain.run(question))
```

🔄 I would have to say the United States. It is the world's largest producer of wine.

✓ QUESTION TO LLM USING API: Which country is largest in the world?

```
question = "Which country is largest in the world?"

print(llm_chain.run(question))
```

🔄 Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Question: What country is largest in the world?

Answer: Let's think step by step. Russia, Saudi Arabia, Turkey is also the largest, and China is China's largest

✓ QUESTION TO LLM USING API: Which is the most populous country in the world?

```
from langchain import PromptTemplate, HuggingFaceHub, LLMChain

template = """Question: {question}

Answer: Let's think step by step."""

prompt = PromptTemplate(template=template, input_variables=["question"])

llm_chain = LLMChain(prompt=prompt,
                    llm=local_llm
                    )

question = "Which is the most populous country in world?"

print(llm_chain.run(question))
```

↔ You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a dataset
India is the most populous country in the world with a population of over 1.3 billion people.



✓ QUESTION TO LLM USING API: What do you see in the night sky?

```
question = "What do you see in the night sky?"

print(llm_chain.run(question))
```

↔ I see a lot of stars when I'm out at night. Do you see many stars?

✓ QUESTION TO LLM USING API: What do you feel when you see a moon?

```
question = "What do you feel when you see a moon?"

print(llm_chain.run(question))
```

↔ That's a good way to look at it. I feel the same way when I see the moon.

✓ QUESTION TO LLM USING API: What is the most beautiful place in the world?

```
question = "What is the most beautiful place in the world?"

print(llm_chain.run(question))
```

↔ That is a good question. I would have to say the ocean. I love the water.

✓ USE PRETRAINED MODEL API TO SUMMARIZE IOT DATA (HOUSE HOLD POWER CONSUMPTION WITH MIN_POWER AND MAX_POWER FOR 10 DAYS)

DESCRIPTION:

This code loads household power consumption data from a CSV (household_power_clean.csv), then uses a pre-trained language model (specifically, facebook/bart-large-cnn) via Hugging Face Transformers and Langchain to generate daily summaries of the minimum and maximum power usage. It iterates through a date range, extracts the min/max power for each day, formats this data as JSON, and feeds it to the LLM. The LLM then generates a summary for each day, which is printed to the console. Deprecation warnings are addressed by using the updated Langchain imports and methods.

Here are five important APIs from the provided code and search results, specifically focusing on the `HuggingFacePipeline` and related functionalities within the LangChain framework:

1. `HuggingFacePipeline`:

- This API allows users to integrate Hugging Face models into LangChain easily. It supports various tasks such as text generation, summarization, and translation. Users can create an instance of `HuggingFacePipeline` either by specifying a model ID or by passing an existing Hugging Face pipeline.
- Example:

```
from langchain_huggingface import HuggingFacePipeline
hf = HuggingFacePipeline.from_model_id(model_id="gpt2", task="text-generation", pipeline_kwargs={'
```



2. `ChatHuggingFace`:

- This API provides a wrapper for chat-based models from Hugging Face. It allows for conversational interactions with models designed for chat applications. The `ChatHuggingFace` class can be instantiated similarly to `HuggingFacePipeline`.
- Example:

```
from langchain_huggingface import ChatHuggingFace
chat_model = ChatHuggingFace(llm=hf)
response = chat_model.invoke("Can you give summary of minimu power usage?")
```

3. `HuggingFaceEndpoint`:

- This API allows users to interact with Hugging Face models hosted on their endpoints. It is particularly useful for users with pro accounts or enterprise subscriptions, enabling them to leverage serverless APIs for model inference.
- Example:

```
from langchain_huggingface import HuggingFaceEndpoint
endpoint_model = HuggingFaceEndpoint(repo_id="meta-llama/Meta-Llama-3-8B-Instruct", task="text-ger
response = endpoint_model.invoke("Can you generate summary of power consumed?")
```



4. `LLMChain ()`:

- This is for chaining operations in LangChain. It allows users to compose sequences of operations (like prompts and model invocations) in a more flexible manner.
- Example:

```

llm_chain = prompt_template | hf # Using the new RunnableSequence approach
summary = llm_chain.invoke({"data": json.dumps(data_json)})

```

These APIs facilitate various interactions with language models, allowing for text generation, summarization, and conversational capabilities while adhering to the latest practices in the LangChain framework.

```

from transformers import pipeline, AutoTokenizer, AutoModelForSeq2SeqLM
import warnings
from langchain.llms import HuggingFacePipeline
from langchain import PromptTemplate, LLMChain
import pandas as pd
from datetime import datetime, timedelta
import json

# Suppress the specific UserWarning (or FutureWarning, adjust if needed)
warning_category = UserWarning # Or FutureWarning, depending on the exact warning type

# Use context manager to suppress warnings only within this block
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=warning_category)

# 1. Load your data (replace with your actual path)
file_path = '/content/drive/MyDrive/AAI-530-DataAnalyticsAndIOT/household_power_clean.csv' # **REPLACE WITH \

try:
    df = pd.read_csv(file_path)
    df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], format='%Y-%m-%d %H:%M:%S')
    df.set_index('Datetime', inplace=True)
except FileNotFoundError:
    print(f"Error: File not found at {file_path}")
    exit()
except Exception as e:
    print(f"Error loading or processing data: {e}")
    exit()

# 2. Initialize LLM (using a summarization model)
model_id = "facebook/bart-large-cnn" # Or another summarization model

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForSeq2SeqLM.from_pretrained(model_id)

pipe = pipeline(
    "summarization",
    model=model,
    tokenizer=tokenizer,
    max_length=40, # Reduced max_length to be less than input length
    min_length=10, # Added min_length to ensure a reasonable summary length
    truncation=True,
)

llm = HuggingFacePipeline(pipeline=pipe) # Corrected instantiation

# 3. Define summarization function (using LLMChain)
def summarize_power_data(data_json): # Changed input to data_json
    """Summarizes min/max power data for a given day using LLMChain, now expecting JSON input."""

    template = """
    Summarize the following household power data (JSON format):

    """

```

```

{data}
'''
"""
prompt_template = PromptTemplate(input_variables=["data"], template=template)

# Use RunnableSequence instead of LLMChain (addressing deprecation warning)
llm_chain = prompt_template | llm

try:
    summary = llm_chain.invoke({"data": json.dumps(data_json)})
    return summary
except Exception as e:
    return f"Error during summarization: {e}"

# 4. Set date range and loop
start_date = datetime(2006, 12, 16) # Start date
num_days = 10

print(f"SUMMARY OF MIN_POWER AND MAX_POWER FOR 10 DAYS")
for i in range(num_days):
    current_date = start_date + timedelta(days=i)
    date_str = current_date.strftime('%Y-%m-%d')

    try:
        daily_data = df.loc[date_str]
        min_power = daily_data['Global_active_power'].min()
        max_power = daily_data['Global_active_power'].max()


        data_to_summarize = {
            'date': date_str,
            'min_power': round(min_power, 3),
            'max_power': round(max_power, 3)
        }

        # Convert data_to_summarize to JSON format
        data_json = data_to_summarize # Keep the dictionary

        summary = summarize_power_data(data_json) # Pass the dictionary directly
        print(f"\nSummary for {date_str}:\n{summary}")

    except KeyError:
        print(f"No data found for {date_str}")
    except Exception as e:
        print(f"Error processing {date_str}: {e}")

```

 Device set to use cpu
 Asking to truncate to max_length but no maximum length is provided and the model has no predefined maximum length
 SUMMARY OF MIN_POWER AND MAX_POWER FOR 10 DAYS

Summary for 2006-12-16:
 Household power data (JSON format): "min_power": 0.284, "max_power": 7.706 "date": "2006-12-16", "

Summary for 2006-12-17:
 Household power data (JSON format): "min_power": 0.206, "max_power": 7.064 "date": "2006-12-17", "

Summary for 2006-12-18:
 Household power data (JSON format): "min_power": 0.202, "max_power": 6.158 "date": "2006-12-18", "

Summary for 2006-12-19:
 Household power data (JSON format): "min_power": 0.194, "max_power": 7.84, "date": "2006-12-19", "

Summary for 2006-12-20:
 Household power data (JSON format): "min_power": 0.202, "max_power": 5.988 "date": "2006-12-20", "

Summary for 2006-12-21:

Household power data (JSON format): "min_power": 0.194, "max_power": 5.854" "date": "2006-12-21",

Summary for 2006-12-22:

Household power data (JSON format): "min_power": 0.202, "max_power": 7.884" "date": "2006-12-22", "

Summary for 2006-12-23:

Household power data (JSON format): "min_power": 0.208, "max_power": 8.698" "date": "2006-12-23", "

Summary for 2006-12-24:

Household power data (JSON format): "min_power": 0.196, "max_power": 6.824" "date": "2006-12-24",

Summary for 2006-12-25:

Household power data (JSON format): "min_power": 0.198, "max_power": 6.702" "date": "2006-12-25", "

✓ CONCLUSION:

The usage of APIs from pre-trained Large Language Models (LLMs) in coding has become increasingly significant as developers leverage these models for various tasks. Here's a summary based on the search results:

1. Code Completion and Generation:

- Pre-trained LLMs, such as Codex and GPT variants, are widely used for generating code completions. These models can suggest code snippets, functions, or entire classes based on the context provided by developers. They are particularly effective in Integrated Development Environments (IDEs) and code editors, enhancing productivity by reducing the amount of boilerplate code that developers need to write.

2. Handling Deprecated APIs:

- LLMs struggle with using up-to-date APIs due to rapid library evolution. This can lead to the generation of code that includes deprecated API calls. To address this, lightweight mitigation strategies like `ReplaceAPI` and `InsertPrompt` have been proposed to help LLMs adapt to changes in APIs by replacing deprecated usages with their current counterparts during the code generation process.

3. Interfacing with LLM APIs:

- When using LLM APIs, applications typically send HTTP requests formatted in JSON, which include parameters such as model variant and prompts. The API processes these requests and returns responses generated by the LLM, which can be used for tasks like content generation, question answering, or sentiment analysis.

4. Fine-tuning Pre-trained Models:

- Developers can fine-tune pre-trained LLMs on specific code libraries or datasets to improve performance on niche tasks. This involves training the model further on a smaller dataset tailored to the specific programming language or framework being used.

5. Integration and Usability:

- LLM APIs bridge the gap between complex AI models and practical applications by providing a straightforward interface for developers. This usability is critical for integrating advanced AI capabilities into existing software systems without requiring deep expertise in machine learning.

Overall, pre-trained LLM APIs facilitate various coding tasks by providing intelligent code suggestions, handling deprecated API issues, and enabling easy integration into development workflows. These capabilities significantly enhance developer efficiency and reduce errors in software development.