

A Report on Recent Applications and Frameworks Using Large Language Models

Muhammad Roman, Barry Devereux

EEECs

Queen's University Belfast

Belfast, UK

{m.roman, b.devereux}@qub.ac.uk

September 2023

Contents

1	Introduction	2
1.1	Base LLMs	2
1.2	Fine-tuned and Instruction-tuned LLMs	3
2	Prompt Designing and Engineering	8
2.1	Basic Prompting	8
2.2	Instruction Prompting	10
2.3	Chain-of-Thought (CoT)	12
3	LLM Frameworks	13
3.1	LangChain	13
4	Retrieval Augmented Generation (RAG)	14
4.1	Using Large Language Models on Private Data: A Case Study with LangChain and LLaMA 2	15
4.1.1	Loading Multiple Documents from a Directory	18
5	Other Applications and Methods	19
A	Appendix A. Using Large Language Models on Private Data: A Case Study with LangChain and LLaMA: technical implementation	23
A.1	Initialisation	23
A.2	Implementing Hugging Face Pipeline	23
A.3	Document Loading	23
A.4	Splitting Document into Chunks	24
A.5	Creating Embedding and Storing in Vector Storage	24
A.6	User Queries and Semantic Search	24

B	List of Documents Used	24
B.1	Energy for a Better Future	24
B.2	Powering Progress	24
B.3	Powering Progress	25
B.4	Powering Progress	25
B.5	Our Non-financial Scorecard	25
B.6	PricewaterhouseCoopers LLP	25

1 Introduction

Large Language Models (LLMs) are machine learning models that are trained on a very large amounts of data from various sources, including books, articles, and web pages. LLMs are a type of artificial intelligence (AI) algorithm that use deep learning techniques to understand, infer, predict, and generate natural language. With the advent of the Transformer deep learning architecture (Vaswani et al., 2017), and commercial systems trained using it, LLMs have shown remarkable emergent behaviors and can generate natural language output on the basis of a prompt and their training data in a way that mimics human abilities.

The strength of any LLM lies in the size and type of the dataset used for its training. The model’s strengths and weaknesses can be explored by testing a model on various tasks, including question/answering, summary generation, sentiment analysis, named entity recognition, chat dialogs, and other standard NLP tasks. In particular models can be compared using various benchmark test datasets and metrics, including the MMLU, BoolQ, Natural Questions, and TriviaQA metrics, each of which tests a model from different perspectives.

To further improve a model’s knowledge in a particular domain, e.g., health sciences, LLMs can be fine-tuned on a smaller set of data from that specific domain. Fine-tuning can be challenging regarding their complexity, time, relevant data, and interpretation, as these models are just black boxes. In a complementary approach, larger-sized models can be prompt-engineered to achieve better results on specific tasks or specific domains. The models can also be provided context using some external context in the form of text or additional documents to the model as part of prompt engineering.

In this report, we survey the recent literature and developments in the world of LLMs, and discuss various kinds of functionality that can be built on top of LLMs to solve complex and challenging real-world tasks.

1.1 Base LLMs

LLMs can be mainly partitioned into two types: base or “foundation” models trained on a very general dataset and with a fundamental language modelling objective (i.e. predicting the next word or token) and fine-tuned and instruction-tuned LLMs that are adapted to a more specific domain or use case.

Base models are typically trained on very large text corpora from the Internet, such as the CommonCrawl¹. The goal of these foundation models is to learn the statistical regularities of language, including the syntactic structure and the semantics of words and phrases, as this kind of information must be learned in order to solve the language modelling objective of predicting the next token of input (or masked tokens within the input). The primary goal is to model the statistical regularities of the language input as generally and as comprehensively as

¹<https://commoncrawl.org>

possible, using this simple token-prediction objective, rather than solve any particular Natural Language Processing (NLP) task (as has historically been the approach in NLP engineering). Once trained, *transfer learning* can be used to apply such models to various tasks, with more task-relevant objectives. In this way, base models can be applied to a variety of problems, including text completion, generation, summarization, and more. However, base models may not always produce outputs perfect answers in specialized domains as these models are not trained on a task-specific corpus. Some important base models include BERT, GLaM, PaLM, BLOOM, GTP-3, LaMDA, YaLM, LLaMa, Falcon, GPT-J, and GPT-Neo.

1.2 Fine-tuned and Instruction-tuned LLMs

Unlike base models, fine-tuned models are trained on more specific data such as scholarly articles from a particular research domain, medical databases, legal texts, and so on. A fine-tuned model can perform better than a base model in that specific domain. Also, these models can sometimes lose general language abilities learned during base model pretraining (depending on the characteristics of the fine-tuning data) and they may find different meanings to some pre-learned concepts in a data-specific context. Fine-tuned models are more specialized and optimized for a special task.

Instruction-tuned models are more versatile and adaptable to multiple tasks. These models try to follow the instructions that are provided. They are fine-tuned on instructions and sample good attempts to follow those instructions. These models are often further refined using Reinforcement Learning from Human Feedback (RLHF), which trains the model by reward and punishment for their good and bad responses by taking feedback from humans. These models are ideal for applications that require generating responses or content based on specific prompts, guidelines, or instructions. Some of the instruction-tuned LLMs include Instruct-GPT, Orca, Alpaca, BLOOMZ, Flan-PaLM, and OpenChatKit. Some famous RLHF fine-tuned LLMs include GPT 4, Sparrow, and Claude. Many foundation LLMs have instruction-tuned versions available for use. For example, ChatGPT is a chat model fine-tuned with RLHF for chat functionality and is based on OpenAI’s GPT3.5 series of large language models.

Table 1 provides a list of large language models that are frequently used by the research and development community.

Table 1: List of prominent state-of-the-art large language models in the order of release date.

Model Name	Developer	Params	Corpus	License	Comments
BERT (Devlin et al., 2019)	Google	340M	3.3B	Apache 2.0	An encoder-only transform-based model, further fine-tuned for more specific tasks include RoBERTa (Liu et al., 2019), ScieBERT, BioBERT (J. Lee et al., 2020), and DistilBERT (Sanh et al., 2019).
XLNet (Yang et al., 2019)	Google	340M	33B	Apache 2.0	Designed as encoder only by Google.

GPT-2 (Radford et al., 2019), GPT-3 (Brown et al., 2020), GPT-4 (OpenAI, 2023)	OpenAI ²	1.5B, 175B, 1.7T	10, 300B	Proprietary	GPT-3.5 and GPT-4 are available to the public via ChatGPT ³ . Also accessible through API with an OpenAI account.
Falcon (Sakor et al., 2020)	Technology Innovation Institute	40B	1T	Apache 2.0	Trained by tokens from RefinedWeb and some curated corpora. Training cost was around 2700 petaFLOP-days, 75% that of GPT-3.
GPT-Neo (Black et al., 2021; Gao et al., 2020)	EleutherAI	2.7B	825 GB	MIT	A GPT-3 alternative, which can perform well in comparison to the smallest endpoint of GPT-3.
GPT-J (Wang & Komatsuzaki, 2021)	EleutherAI	6B	825 GB	Apache 2.0	GPT-3-style language model. Because of its free and open usage, it gained much interest. Proved to be better than GTPNeo in various benchmarks.
Gopher (Rae et al., 2021)	DeepMind	280B	300B	Proprietary	Evaluated on 152 diverse tasks, achieving state-of-the-art performance across the majority. Best performance is observed in tasks related to reading comprehension, fact-finding, and bullying detection. Does not perform well in mathematical reasoning.
Ernie 3.0 Titan (D. Zeng et al., 2021)	Baidu	260B	4TB	Proprietary	Chinese-language LLM. Ernie Bot is based on this model. Recently upgraded for Baidu bot in 2023.
GPT-NeoX (Black et al., 2021)	EleutherAI	20B	825GB	Apache 2.0	A suite of 6 diverse tasks based on the Megatron architecture.
PanGu (W. Zeng et al., 2021)	Huawei	1.085T	329B	Proprietary	Trained on large Chinese datasets with a focus on knowledge capabilities.

²OpenAI 2015, <https://openai.com>

³ChatGPT 2022 <https://www.anthropic.com>

Megatron-Turing NLG (Smith et al., 2022)	Microsoft and Nvidia	530B	338.6B	Restricted web access only	Achieved superior zero-, one- and few-shot learning accuracy on several NLP benchmarks and established new state-of-the-art results.
GLaM (Du et al., 2022)	Google	1.2T	1.6T	Proprietary	Scales the model capacity while incurring substantially less training cost compared to dense variants. Achieved better overall few-shot performance across 29 NLP tasks.
Claude (Anthropic, 2022)	Anthropic	52B	400B	Closed	Fine-tuned for desirable behavior in conversations. Beta Claude Instant costs \$ 1.63/M tokens to prompt and \$ 5.51/M tokens on completion.
LaMDA (Thoppilan et al., 2022)	Google	137B	1.56T words, 168B tokens	Proprietary	Specialized for response generation in conversations.
Chinchilla (Hoffmann et al., 2022)	DeepMind	70B	1.4T tokens	Proprietary	Reduced-parameter model trained on more data. Used in the Sparrow bot.
PaLM (Chowdhery, Narang, et al., 2022)	Google	540B	768B	Proprietary	Aimed to reach the practical limits of model scale
OPT (Z. Zhang et al., 2022)	Meta	175B	180B	Non-commercial research	GPT-3 architecture with some adaptations from Megatron
YaLM 100B (Hwang et al., 2022)	Yandex	100B	1.7TB	Apache 2.0	English-Russian model based on Microsoft’s Megatron-LM.
Minerva (Lewkowycz et al., 2022)	Google	540B	38.5B	Proprietary	Based on the PaLM model. Trained on mathematical and scientific data using arXiv preprints; aims to solve mathematical and scientific questions using step-by-step reasoning.

BLOOM (Lu et al., 2022)	Academic collaboration led by Hugging Face	175B	350B (1.6TB)	Responsible AI Licence	Essentially GPT-3 but trained on a multi-lingual corpus (30% English excluding programming languages)
Galactica (Taylor et al., 2022)	Meta	120B	106B	CC-BY-NC-4.0	Trained on scientific text and modalities.
AlexaTM (Soltan et al., 2022)	Amazon	20B	1.3T	Proprietary	Bidirectional sequence-to-sequence architecture. It outperformed GPT3 (175B) on SuperGLUE and SQuADv2 datasets and provided SOTA performance on multilingual tasks such as XNLI, XCOPA, Paws-X, and XWinograd.
LLaMA (Kipf et al., 2022)	Meta	65B	1.4T	Non-commercial research	Trained on a large 20-language corpus to aim for better performance with fewer parameters. Researchers from Stanford University trained a fine-tuned model based on LLaMA weights called Alpaca.
PaLM-2 (Chowdhery, Kim, et al., 2022)	Google	340B	3.6T	Proprietary	Used in Google Bard chatbot
Orca (Han & Tsvetkov, 2022)	Microsoft	13B	Synthetic data from GPT-4	Non-commercial	Descendant of LLaMA
Vicuna-13B (H. Zhang et al., 2022)	LMSYS	370M	-	Non-commercial license	Fine-tuning LLaMA on user-shared conversations collected from ShareGPT
Pythia (Conover et al., 2023)	EleutherAI	12B	-	Apache-2.0 license	A suite of 16 LLMs all trained on public data.
Cerebras-GPT (Dey et al., 2023)	Cerebras	13B	-	Apache 2.0	Trained with Chinchilla formula.

Bloomberg GPT (Wu et al., 2023)	Bloomberg L.P.	50B	708B	Proprietary	Provides a good performance on general LLM benchmarks while specialized in financial tasks. Based on Bloomberg's extensive data sources, the model is trained on a 363 billion token dataset, augmented with 345 billion tokens from general-purpose datasets.
OpenAssistant (Kopf et al., 2023)	LAION	17B	1.5T	Apache 2.0	A chat-based assistant, trained on crowdsourced open data, which can interact with third-party systems to retrieve information dynamically.
Jurassic-2 (Labs, 2023)	AI21 Labs	178B	-	Proprietary	Trained on dialog data and optimized for conversational AI.
GPT4All (Anand et al., 2023)	Nomic AI	7B	-	Apache-2	Open and free to use. It can be installed locally and fine-tuned. Can work with LangChain framework.
GPT4All-J (Yuvanesh et al., 2023)	Nomic AI			Apache-2	Fine-tuned from GPT-J on assistant style interaction data. It can be downloaded locally.
Phi-1 (Gunasekar et al., 2023)	Microsoft	350M	7B	-	Specialized for Python coding task. Trained on textbook quality data, synthetic generation from GPT-3.5, and web-sourced filtering.
LLaMA 2 (Kipf et al., 2023)	Meta	70B	2T	Open Access	Successor of LLaMA. It has doubled the context length of LLaMA 1 to 4096. Its chat is safe, based on evaluation by human raters using 2,000 adversarial prompts.
Dolly (Conover et al., 2023)	Databricks	12B	-	Licensed for commercial use	A causal language model derived from Pythia model family.

Platypus (A. N. Lee et al., 2023)	Boston University	70B	-	Open Access	The Platypus models are a series of fine-tuned and merged variants based on the LLaMA and LLaMa-2 transformer architectures. Platypus takes advantage of LoRA (Hu et al., 2022) and PEFT (Mangrulkar et al., 2022) for fine-tuning, making it a top-ranked open small model on Hugging Face LLM Leaderboard in spite of its smaller size.
Falcon 180B (Philipp et al., 2023)	Technology Innovation Institute	180B	3.5T	Free for Personal/Research only	Despite having a smaller size in comparison, it has comparable performance to Google's PaLM 2 (Bard) and is not far behind GPT-4 while outperforming GPT-3.5 on MMLU. It is ranked as the highest-performing pretrained LLM on the Hugging Face Open LLM Leaderboard ⁴ . Its large size requires large memory for training, 640GB-5120GB.

2 Prompt Designing and Engineering

Depending upon the type of tasks we wish to solve, there are various ways to prompt the LLMs. Prompt engineering refers to methods of communicating with the LLMs without updating the weights of a model. The effects of prompt engineering (i.e. the design of the prompt text) can affect the outcome of a model.

2.1 Basic Prompting

This prompting is the simplest way of instructing an LLM to solve a given task and is often used for benchmarking. There are two types of basic prompting:

Zero-Shot In zero-shot prompting we attempt to use a model's existing knowledge based on its learning to make predictions of some unseen data without any guidance in the instruction given to it. In this type of prompting, we feed the text of a task to a model and ask for the associated type of answer.

⁴https://huggingface.com/spaces/HuggingFaceH4/open_llm_leaderboard

Input: Together, we have transformed our company's competitive position and led the way towards a safer, smarter, more sustainable future that delivers enduring value for all our stakeholders, and our planet.^a

Sentiment:

^aPWC Annual Report 2021

The LLM answer for this would be:

Sentiment: Positive

In this example, the model could identify the sentiment of the input sentence without any instructions given to the model for sentiment classification (relying entirely on the meaning of the word "sentiment" as learned in the model's training). However, in some cases, the model cannot provide results without having some examples/samples to learn from.

Few-shot In few-shot prompting, a model is given a number of samples for the model to adapt itself to the nature of the problem at hand and to relate the input and output pairs. As the model first observes good examples, it can better understand human intention and criteria for desired answers. Therefore, few-shot learning often gives better responses than zero-shot. However, it comes at the cost of more token consumption and may hit the context length limit when the input and output text are long.

Consider the following few-shot example:

Examples:

Input: "Together, we have transformed our company's competitive position."

Sentiment: Positive

Input: "After years of collaboration, we've reshaped our company's trajectory."

Sentiment: Positive

Input: "Through joint efforts, we've revolutionized our corporate standing."

Sentiment: Positive

Input: "Despite our best efforts, we've encountered challenges that mar our progress."

Sentiment: Negative

Input: "Despite previous success, recent setbacks have cast a shadow over our achievements."

Sentiment: Negative

Input: "Our journey has been marred by difficulties, impacting our ability to deliver value."

Sentiment: Negative

Task:

Provide the sentiment for the following statement.

"Our dedication, specialty in the financial field, and leveraging our staff with state-of-the-art training have led us to a new benchmark in the financial sector this year."

Response:

Sentiment: Positive

The model is given a number of examples to learn from and then solves the task of classifying a statement as Positive or Negative based on those examples.

2.2 Instruction Prompting

Instruction prompting is a technique where explicit and detailed instructions are given to guide the language model in generating specific content. The ultimate goal is to align the model answer more with a user's intention. Consider the following example where the LLM is instructed with some directions and prompted with user queries. The prompt includes substantial task relevant information:

Three Roles and Responsibilities

The Bank's Three Lines of Defence (3LOD) governance model is designed to ensure appropriate responsibility and accountability is allocated to the management, reporting and escalation of risks. Fair treatment of vulnerable customers must be embedded into the Bank's culture.

First Line of Defence (LOD)

Responsibilities:

1. Implementing this guidance; providing appropriate staff training and creating and maintaining processes and procedures to ensure that vulnerable customers are identified and appropriate action is taken to demonstrate good customer outcomes
2. Ensuring that proportionate oversight (e.g. quality assurance) activity is undertaken in line with the RMF and the effectiveness of the systems and controls are assessed to confirm that they continue to be effective.
3. All staff who are interacting with customers should periodically and at least annually, receive training to identify vulnerability. This training can be tailored to their role and should signpost to tools that the adviser can access to help them further. A culture of support for customers and staff must be maintained at all times with a clear focus on treating the customer fairly, ensuring that they receive a similar outcome as other customers.

Second Line of Defence (The Risk Function)

Responsibilities:

1. Providing technical advice and guidance to 1LOD to support the implementation of this policy guidance.
2. Independent oversight of the systems and controls that have been put in place.

Third Line of Defence Risk Framework Owner (RFO)

Vulnerable Customer is an important element of Conduct Risk and the RFO in the 2nd LOD is the Director of Compliance. The RFO for Conduct Risk is responsible for:

1. Upkeep of the Risk Policy and necessary Control Standards
2. Defining risk appetite and measurement
3. Implementing appropriate oversight and assurance^a

Task: NONAME Bank has listed no training schedule in its annual report. Which line of action does it violate?

^aPWC Risk Policy Overview – Vulnerable Customer 2023 Document

The output using ChatGPT is as follows:

Response:

The situation described violates the responsibilities of the "**First Line of Defence**" (1LOD) in the Three Lines of Defence (3LOD) governance model. Specifically, it violates the following responsibility of the First Line of Defence:

Responsibility Violated:

3. *All staff who are interacting with customers should periodically and at least annually, receive training to identify vulnerability. This training can be tailored to their role and should signpost to tools that the adviser can access to help them further.*

In this case, NONAME Bank's failure to list any training schedule for identifying vulnerability in its annual report means that it is not fulfilling its responsibility to ensure that staff members interacting with customers are periodically trained to identify vulnerability. This training is crucial to identifying and supporting vulnerable customers as well as ensuring that customers are treated fairly and receive appropriate outcomes.

Therefore, the violation falls under the responsibility of the "First Line of Defence" within the 3LOD governance model.

Such a context-specific answer cannot be obtained with a zero or few-shot prompting as instructional prompting provides more contextual information as part of the query to the LLM.

2.3 Chain-of-Thought (CoT)

In this type of prompting, a series of intermediate reasoning steps, also called reasoning chains, are provided to lead to the outcome. It helps in the case of more complicated tasks where the answer needs step-wise explanation, for Example (Chen et al., 2023);

Question: Q: A tower is made out of 4 blue blocks, twice as many yellow blocks, and an unknown number of red blocks. If there are 32 blocks in the tower in total, how many red blocks are there? A:

Text-Davinci-002 Answer: There are 8 red blocks in the tower. ✗

Text-Davinci-003 Answer: There are 16 red blocks in the tower. ✗

ChatGPT Answer: Let's start by using algebra to solve for the unknown number of red blocks. Let R be the number of red blocks. We know that there are 4 blue blocks and twice as many yellow blocks, which means there are $2 \times 4 = 8$ yellow blocks. So the total number of blocks in the tower is: 4 (blue) + 8 (yellow) + R (red) = 32. Simplifying this equation: $12 + R = 32$. Subtracting 12 from both sides: $R = 20$. Therefore, there are 20 red blocks in the tower. ✓

It can be seen from the example above that GPT-3's (Text-Davinci-002 and Text-Davinci-003) answer is wrong. Some instruction fine-tuned models, such as ChatGPT, are trained with CoT prompting, which implicitly converts such complex tasks into a chain of thoughts and correctly answers the question in the example above. An improvement from 17.7% to 78.7% has been reported in GPT-3's accuracy by simply adding the instruction 'Let's think step-by-step' to each input query of MultiArith dataset (Chen et al., 2023). Such a type of CoT prompting is called a zero-shot CoT prompt. In the case of a few-shot CoT, each step of breaking the

problem is instructed to make the model understand how to break the problem and then solve it piece-by-piece, for example:

Question: How many keystrokes are needed to type the numbers from 1 to 500? Answer
Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788
Reasoning: There are 9 one-digit numbers from 1 to 9. There are 90 two-digit numbers from 10 to 99. There are 401 three-digit numbers from 100 to 500. $9 + 90(2) + 401(3) = 1392$.
Answer: (b)

3 LLM Frameworks

LLM frameworks provide tools and features designed to help developers of applications leverage the power of LLMs. They use an underlying LLM of our choice and provide access to them through various prompting methods. We can also contextualize our prompts with instructions and external documents, e.g., agency documents, books, and websites. LLM frameworks abstract away the complexity of the backbone LLM models from application developers.

3.1 LangChain

LangChain is a framework that provides access to a set of libraries, tools, and features to access large language models to make AI-driven applications. Some of the main features of LanChain include:

LLMs Integration This allows the integration of large language models and leverages their capabilities for tasks including language generation, understanding, inference, summarization, and sentiment.

Prompts LangChain provides a variety of prompting styles and enables the developers to engineer their own prompt templates and perform prompt chaining.

Memory This functionality allows the system to store and retrieve messages during a conversational dialog for later use.

Indexes Indexing provides a way of structuring documents for LLMs. The module helps in providing different types of functions to work with documents and their context for chaining the indexes. The most important task performed using indexes is relevant document retrieval, where we provide a query and the model returns a relevant document. The classes included in this module are related to document loading, text splitting, vector stores, and retrievers.

Agents Agents are autonomous tools that make decisions on their own to look for external resources and decide on the best course of action to achieve a specific goal or output.

In addition to LangChain, other frameworks and tools can be used to build applications around LLMs. Some of them are listed below:

1. Auto-GPT

2. BabyAGI
3. AgentGPT
4. SuperAGI
5. Stack AI
6. Semantic
7. LangSmith
8. Langdock
9. GradientJ
10. Arches AI
11. Azure OpenAI Service
12. Steamship
13. Deep AI
14. Quarkus
15. PromptLayer
16. Vite
17. Giotto
18. Flowise

4 Retrieval Augmented Generation (RAG)

LLMs have demonstrated impressive capabilities for a wide range of tasks; however, there are a few important constraints resulting from how they are trained:

- i. The cut-off time:** LLMs are trained on a particular corpus, which is up-to-date to a given time. For example, ChatGPT, utilising GPT-4 as the language model, has a data cut-off of September 2021. New information after the cut-off time is not available to the model. For example, if you ask the question, “Who is the prime minister of the UK?”, the model responds that Boris Johnson is the most recent PM based on its available data. Since training base language models is enormously computationally expensive, they are not typically retrained often with more up-to-date datasets.
- ii. Parametric memory:** During pre-training, LLMs establish a so-called ‘Parametric Memory’ stored in the neural weights. ‘Parametric Memory’ is the knowledge a model learns during its initial training, and, not unlike human semantic memory, this knowledge is a synthesis of everything that has been learned during training – it does not involve the look-up of specific memorized example documents. A model responding to queries through its parametric memory’ therefore usually lacks the citation of the source of information. Users would often prefer that sources of information be cited explicitly and that the output of the model be connected to relevant supporting sources.
- iii. Generation of private/corporate data:** Businesses sometimes require knowledge to be generated from private organizational data, whereas LLMs provide text based on the general-purpose corpora.

Some of these problems can be solved with Retrieval Augmented Generation (RAG). This technique allows the augmenting of a prompt with some additional context, usually a document retrieved from an external source. This supplements the ‘Parametric Memory’ by contextualizing the prompt with the most relevant data.

LangChain provides components to support the concept of RAG and provides context with a prompt for a fine-tuned task.

4.1 Using Large Language Models on Private Data: A Case Study with LangChain and LLaMA 2

Despite the gigantic corpus size used with language models in pretraining, most businesses run on local private data, which is usually not accessible for training the base models. Leveraging LLMs with corporate and private data can provide better solutions within that organization’s domain. We have already discussed that base models can be finetuned on this private and specialized data type. We have also explored ways to provide private data as context with input queries and further instruct the LLMs to base their answers on the provided context. RAG builds on these concepts.

This section provides a walkthrough of developing a Question/Answer tool with Hugging Face transformers for the LLaMA 2 pretrained chat model with an external context. A detailed workflow is shown in Figure 1 and explained in the next section.

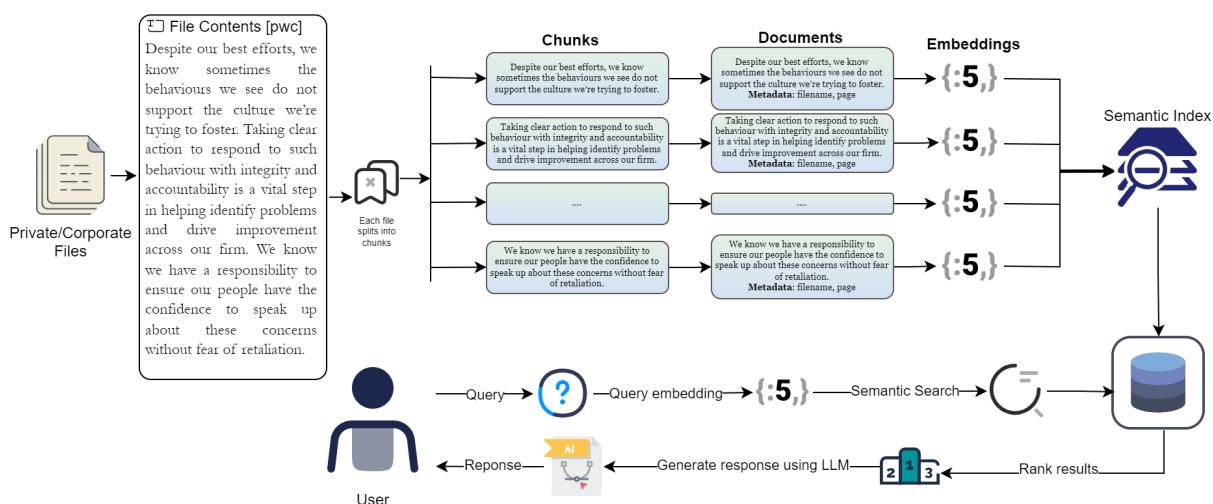


Figure 1: A workflow for using large language models on private data and answering users’ queries from the private data.

To use the LLaMA 2 model with the LangChain framework, we need to setup a number of things, including the computing resources, the transformers, accessing Python libraries, and model weights. Given the size of the model, a T4 GPU was selected as the Hardware Accelerator. Refer to the provided IPython notebook⁵ (and Appendix A) for the Python code implementation. Refer to Figure 1 for an overview of the implementation.

In the implementation, the instantiated chain provides an end-to-end conversation with the vector storage to find relevant information from the provided documents. The LLM used by the chain will generate a text response based on the returned relevant information to the application

⁵/path/on/arc/gdrive

user. A history object can optionally be managed to keep a chain of prompts and provide contextual responses to the user based on their previous questions from the LLM.

The following are some of the queries submitted to LLaMA 2 using Langchain and their responses:

```
# Prompting Case 1
chat_history = []
query = "What is summary of this document?"
result = chain({"question": query, "chat_history": chat_history})
print(result['answer'])
```

RESPONSE: This document outlines the bank's approach to managing vulnerable customers, including the identification and management of conduct risks associated with these customers. It defines the roles and responsibilities of various lines of defense (1st, 2nd, and 3rd) and provides guidelines for embedding a culture of good customer outcomes. Additionally, it sets out the requirements for reporting and resolution of breaches of risk appetite or policy.

```
# Prompting Case 2
chat_history = []
query = "What are the triggers and flags to identify vulnerable customers?"
result = chain({"question": query, "chat_history": chat_history})
print(result['answer'])
```

RESPONSE:
According to the policy, the following are some of the triggers or flags to identify vulnerable customers:

- Long-term underlying illness
- Physical or mental disability
- Bereavement of family member or close friend
- Struggling with financial matters/obligations
- Loss of income
- Large amount of debt
- Difficulty understanding simple financial matters
- Difficulty retaining information, including over multiple interactions
- Relying on a third party for advice
- Avoiding certain types of customer platforms, indicating lack of skills in that area
- Difficulty accessing digital channels

The response is generated based on the similarity of the question embedding to the document chunk embedding in the vector store. To see which document chunks were used to generate this response, we can get it as follows:

```
print(result['source_documents'])
```


page content: 'It is also recognised that all of our customers sit within a spectrum of vulnerability. Within this spectrum, some customers are at greater risk of harm than others. Those customers disclosing or displaying any or multiple drivers of vulnerability are likely to require more support. Early action is vital to prevent and reduce the risk of harm to our customers.2.3.2 Triggers of vulnerability can be revealed within conversations with our customers. These should serve as flags to the advisers, who should then ask searching questions to uncover the full extent of the vulnerability.flags in conversation could include disclosure of: • Long term underlying illness; • Physical or mental disability; • Bereavement of family member or close friend; • • • Struggling with financial matters/obligations; • In financial difficulty; • Business restructure.of income; Large amount of debit;'

source document: 'VulnerableCustomerPolicyOverview2023.pdf'

page content: 'All staff who are interacting with customers should periodically and at least annually, receive training to identify vulnerability. This training can be tailored to their role and should signpost to tools that the adviser can access to help them further.culture of support for customers and staff must be maintained at all times with a clear focustreating the customer fairly, ensuring that they receive a similar outcome as other customers.3.2 Second Line (The Risk Function)second line of Defence (2LOD) are responsible for:technical advice and guidance to 1LOD to support the implementation of this policy guidance. Independent oversight of the systems and controls that have been put in place.3.3 Risk Framework Owner (RFO)'

sourcedocument:VulnerableCustomerPolicyOverview2023.pdf'

page content: 'Other flags could be discovered in conversation with the customer, such as: • Difficulty understanding simple financial matters; • Difficulty retaining information – including over multiple interactions; • Relying on a third party for advice; • Avoidance of certain types of customer platforms, indicating lack of skills in that area –e.g., consideration, sympathy and flexibility should be key tools in handling the initial discussions.care should be taken if several of the flags overlap, as this would indicate a vulnerability that may require extensive and or external support to achieve a good customer outcome.Help and Support The Bank works with external organisations like Citizens Advice and have developed a referral programme for customers who are experiencing difficulties. Get Free Financial Help from Citizens Advice | The Co-operative Bank'

source document: 'VulnerableCustomerPolicyOverview2023.pdf'

page content: "Risk Policy Overview – Vulnerable Customer2023: PUBLIC11.1 Aim-saim of this document is to outline the steps that need to be undertaken to ensure that The Co- operative Bank plc (The Bank) is considering and supporting customers during times of ongoing or temporary vulnerability.is a key focus area for the Financial Conduct Authority (FCA), and by supporting our vulnerable customers we are also aligning to the Co-operative Bank's Values and Ethics, ensuring that our customers are treated fairly.FCA expects the fair treatment of vulnerable customers to be embedded in our culture, processes and policies. In order to do so, this guidance must be adhered to in all areas of The Bank, including any processes that may be handled on our behalf by third parties.

source document:'VulnerableCustomerPolicyOverview2023.pdf'

These four chunks were the most relevant based on the embedding similarities of chunks and query in the vector store. The chunks may be matched from different files, although in this case, the four chunks are all retrieved from the same relevant file. For some other queries, it can generate a response from various files depending on the similarity of embeddings. This can

help build a better response to a prompt.

Prompting Case 3

```
query = "Which trigger or flags is the following statement related to  
'I am having a non-stop working routine for the last 18 hours to pay my  
mortgage and cannot focus any further on the questions being asked by  
you on phone call'"
```

This statement relates to the flag of 'Struggling with financial matters/obligations.'

The statement within the single triple quotes is evaluated against the triggers and flags of vulnerability, and then the particular class of vulnerability is extracted from within the documents. The models could correctly identify the trigger type that defines this type of vulnerability, which can be confirmed from Section 2.3.2 of the 'Risk Policy Overview - Vulnerable Customer 2023' document.

You can also find some more examples of prompts in our shared IPython Notebook.

4.1.1 Loading Multiple Documents from a Directory

In the previous section, we illustrated prompting from a single file. We can also load multiple documents from a directory and then prompt the LLM to look for answers from all the files in the directory. This can be done by using a different document loader from LangChain, e.g., PyPDFDirectoryLoader. By loading multiple files together, we can prompt queries like comparing information from multiple sources. Please refer to our shared IPython notebook for Python implementation. Following are some of the prompt results based on a number of files:

```
query = "What is the Shell annual revenue for each year?"  
result = chain({"question": query, "chat_history": chat_history})
```

The Shell annual revenue for each year is as follows:

- * 2020: \$180,543 million
- * 2019: \$344,877 million
- * 2018: \$388,379 million
- * 2017: \$305,179 million
- * 2016: \$233,591 million

Sources:

```
'/content/pwc/shell-annual-report-2020.pdf', 'page': 327  
'/content/pwc/shell-annual-report-2022.pdf', 'page': 224  
'/content/pwc/shell-annual-report-2021.pdf', 'page': 349  
'/content/pwc/shell-annual-report-2019.pdf', 'page': 55
```

As explained earlier, the page number mentioned in the sources is the virtual page, also called the chunk number. In the next prompt, we ask the model a simple question. The models looks through all chunks in all the files to generate an answer to the question.

```
query = "Who was the Chief Executive Officer of Shell in 2019?"  
result = chain({"question": query, "chat_history": chat_history})  
print(result['answer'])
```

According to the text, the Chief Executive Officer of Shell in 2019 was Ben van Beurden.

Although basic and simple, these examples support our confidence in using the RAG approach for supplying a context in the form of files/directories to LLM's and asking relevant questions without the need to fine-tune the model for those documents. Eliminating and measuring hallucination is a challenge in an RAG pipeline which requires the attention of the researchers.

5 Other Applications and Methods

Beyond the example use cases described in this report, we can also apply a multi-agent-based LLM technique. An autonomous LLM agent can perform one or more actions, all leading towards a solution to a common problem. Chain of thought reasoning can be used to decompose the task into sub-tasks, and then each task is handed over to LLM agents. These agents make autonomous decisions in solving the problem at hand, possibly including a human-in-the-loop.

It is also observed that the documents have data in multiple modalities, including natural text, charts/graphs, images, and tabular data. A single model cannot perform well for all these types of data. Therefore, a viable solution may be to use a multi-model approach using different models for each type of data within the documents, combining models that process text and tables, for example. Furthermore, the multi-agent approach can be potentially utilized to handle each of these models and then combine the information to generate a response to the end user.

Moreover, a fine-tuned model will outperform the general-purpose models that we have examined so far for a specialized task. Fine-tuning large language models on limited computing resources is a big challenge. Techniques like Parameter Efficient Fine Tuning (PEFT) provided a way to perform modular fine-tuning with optimal resources. Retaining a balance between knowledge from the pre-training and adapting to new specialized tasks can be achieved by PEFT. Low-Rank Parameter, Low-Rank Adaptation (LoRA) (Hu et al., 2021) and Quantized Low-Ranking Adaptation (QLoRA) (Dettmers et al., 2023) are some of the recent techniques used for fine-tuning using optimized resources. In the future, we can investigate how these techniques can help us achieve better performance and response to the end users.

References

- Anand, Y., Nussbaum, Z., Duderstadt, B., Schmidt, B., & Mulyar, A. (2023). Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo.
- Anthropic. (2022). Model card and evaluations for claude models. [chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf](https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf)
- Black, S., Gao, L., Wang, P., Leahy, C., & Biderman, S. (2021). *GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow* (Version 1.0) [If you use this software, please cite it using these metadata.]. Zenodo. <https://doi.org/10.5281/zenodo.5297715>
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

- Chen, J., Chen, L., Huang, H., & Zhou, T. (2023). When do you need Chain-of-Thought Prompting for ChatGPT? [arXiv:2304.03262 [cs]]. <https://doi.org/10.48550/arXiv.2304.03262>
- Chowdhery, A., Kim, J. J., Barham, P., Chung, H. W., Chou, M., Dean, J., Devlin, J., Damos, G., Elsen, E., Fang, S. H., et al. (2022). Palm-2: An open ai system for advanced natural language understanding. *arXiv preprint arXiv:2204.02311*.
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.
- Conover, M., Hayes, M., Mathur, A., Xie, J., Wan, J., Shah, S., Ghodsi, A., Wendell, P., Zaharia, M., & Xin, R. (2023). *Free dolly: Introducing the world's first truly open instruction-tuned llm*. Retrieved September 8, 2023, from <https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm>
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient Finetuning of Quantized LLMs [arXiv:2305.14314 [cs]]. <https://doi.org/10.48550/arXiv.2305.14314>
Comment: Extended NeurIPS submission
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, May 24). BERT: Pre-training of deep bidirectional transformers for language understanding. Retrieved November 15, 2022, from <http://arxiv.org/abs/1810.04805>
- Dey, N., Gosal, G., Khachane, H., Marshall, W., Pathria, R., Tom, M., Hestness, J., et al. (2023). Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*.
- Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. (2022). *International Conference on Machine Learning*, 5547–5569.
- Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. (2020). The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.
- Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., et al. (2023). Textbooks are all you need. *arXiv preprint arXiv:2306.11644*.
- Han, X., & Tsvetkov, Y. (2022). Orca: Interpreting prompted language models via locating supporting data evidence in the ocean of pretraining data. *arXiv preprint arXiv:2205.12600*.
- Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. (2022). Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models [arXiv:2106.09685 [cs]]. <https://doi.org/10.48550/arXiv.2106.09685>
Comment: Draft V2 includes better baselines, experiments on GLUE, and more on adapter latency
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2022). LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*. <https://openreview.net/forum?id=nZeVKeeFYf9>
- Hwang, S., Huang, S., Zhao, J., Zhang, O. K., Markert, C., Xiong, W., & Liu, X. (2022). YaLM: Yet another language model.
- Kipf, T., Russell, R., Sablayrolles, A., Escolano, D., Mobahi, H., Zhao, Z., & Houlsby, N. (2023). LLaMA 2.

- Kipf, T., Sablayrolles, A., Escalano, D., Russell, R., Mobahi, H., Zhao, Z., & Houslay, N. (2022). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2205.04725*.
- Kopf, A., Kilcher, Y., von Rutte, D., Anagnostidis, S., Tam, Z.-R., Stevens, K., Barhoum, A., Duc, N. M., Stanley, O., Nagyfi, R., et al. (2023). Openassistant conversations—democratizing large language model alignment. *arXiv preprint arXiv:2304.07327*.
- Labs, A. (2023). Announcing jurassic-2 and task-specific apis.
- Lee, A. N., Hunter, C. J., & Ruiz, N. (2023). Platypus: Quick, cheap, and powerful refinement of llms. *arXiv preprint arXiv:2308.07317*. <https://platypus-llm.github.io>
- Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). Biobert: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4), 1234–1240.
- Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., et al. (2022). Solving quantitative reasoning problems with language models, 2022. URL <https://arxiv.org/abs/2206.14858>.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019, July 26). RoBERTa: A robustly optimized BERT pretraining approach. Retrieved November 15, 2022, from <http://arxiv.org/abs/1907.11692>
- Lu, K., Grover, A., Abrol, S., Chung, M., Kiela, D., Besse, F., Fedus, W., Henriques, D. P., Ho, X. L., Hwang, M., et al. (2022). Bloom: Blueprint for an open llm. *arXiv preprint arXiv:2206.06716*.
- Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., & Paul, S. (2022). Peft: State-of-the-art parameter-efficient fine-tuning methods.
- OpenAI. (2023). GPT-4 Technical Report [arXiv:2303.08774 [cs]]. <https://doi.org/10.48550/arXiv.2303.08774>
Comment: 100 pages
- Philipp, S., Omar, S., Pedro, C., Leandro, W., von, & Julien, L. (2023). Spread your wings: Falcon 180b is here. <https://huggingface.co/blog/falcon-180b>.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*.
- Rae, J., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., et al. (2021). Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.
- Sakor, A., Singh, K., Patel, A., & Vidal, M.-E. (2020). Falcon 2.0: An Entity and Relation Linking Tool over Wikidata. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 3141–3148. <https://doi.org/10.1145/3340531.3412777>
- Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
- Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabh-moye, S., Zerveas, G., Korthikanti, V., Zhang, E., Child, R., Aminabadi, R. Y., Bernauer, J., Song, X., Shoyebi, M., He, Y., Houston, M., Tiwary, S., & Catanzaro, B. (2022). Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model [arXiv:2201.11990 [cs]]. <https://doi.org/10.48550/arXiv.2201.11990>
Comment: Shaden Smith and Mostofa Patwary contributed equally

- Soltan, S., Ananthakrishnan, S., FitzGerald, J., Gupta, R., Hamza, W., Khan, H., Peris, C., Rawls, S., Rosenbaum, A., Rumshisky, A., et al. (2022). Alexatm 20b: Few-shot learning using a large-scale multilingual seq2seq model. *arXiv preprint arXiv:2208.01448*.
- Taylor, R., Kardas, M., Cucurull, G., Scialom, T., Hartshorn, A., Saravia, E., Poulton, A., Kerkez, V., & Stojnic, R. (2022). Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*.
- Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H., Jin, C., Bos, M., Baker, S., Lee, H., et al. (2022). LaMDA: Language models for dialog applications.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Wang, B., & Komatsuzaki, A. (2021). Gpt-j-6b: A 6 billion parameter autoregressive language model. <https://github.com/kingoflolz/mesh-transformer-jax>
- Wu, S., Irsoy, O., Lu, S., Dabrovolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D., & Mann, G. (2023). BloombergGPT: A Large Language Model for Finance [arXiv:2303.17564 [cs, q-fin]]. <https://doi.org/10.48550/arXiv.2303.17564>
Comment: Updated to include Training Chronicles (Appendix C)
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.
- Yuvanesh, A., Zach, N., Brandon, D., Benjamin, S., M., Adam, T., & Andriy, M. (2023). Gpt4all-j: An apache-2 licensed assistant-style chatbot. *Amazonaws*. chrome-extension://efaidnbmninnibpcapcglclefindmkaj/https://s3.amazonaws.com/static.nomic.ai/gpt4all/2023_GPT4All-J_Technical_Report_2.pdf
- Zeng, D., Huang, Y., Liu, Y., Nie, J., Zhao, X., Chen, J., Xiao, Y., Zang, M., Ou, G., Mozafari, M., et al. (2021). Ernie 3.0 titan: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2112.12731*.
- Zeng, W., Ren, X., Su, T., Wang, H., Liao, Y., Wang, Z., Jiang, X., Yang, Z., Wang, K., Zhang, X., Li, C., Gong, Z., Yao, Y., Huang, X., Wang, J., Yu, J., Guo, Q., Yu, Y., Zhang, Y., ... Tian, Y. (2021). PanGu- α : Large-scale Autoregressive Pretrained Chinese Language Models with Auto-parallel Computation [arXiv:2104.12369 [cs]]. <https://doi.org/10.48550/arXiv.2104.12369>
Comment: The technique report for PanGu- α
- Zhang, H., Gao, A., Yamada, H., Tang, P., Yu, S., & Leskovec, J. (2022). Vicuna-13B.
- Zhang, Z., Han, Y., Zhou, H., Ke, P., Yasunaga, M., Chen, S., Wang, Y., Zhang, Z., Awadallah, A. H., Chen, H., et al. (2022). Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

A Appendix A. Using Large Language Models on Private Data: A Case Study with LangChain and LLaMA: technical implementation

A.1 Initialisation

We initialize a text-generation pipeline with Hugging Face for the pretrained LLaMA 2 model. The model is downloaded from Meta AI ⁶ and accessed through Hugging Face.

The text-generation pipeline generates text as output, which requires stopping criteria to stop generating any more text. If the stopping criteria are not defined, the model generates text that may not be relevant to the desired output. The stopping criteria can be a fixed length of output (e.g., stop after 512 tokens), stopping at a particular keyword or phrase, stopping when the model repeats itself, stopping at a logical conclusion or when a question is answered, or in a conversational chat stop when human starts speaking (Cell#5 in the notebook).

A.2 Implementing Hugging Face Pipeline

Once the basic prerequisite settings are accomplished, we are reading to initialize the Hugging Face pipeline. A pipeline is an easy way to use a large language model. The pipeline objects abstract out the complexity of the code by offering API calls to tasks, including question answering, inference, sentiment analysis, feature extraction, masked language modeling, and named entity recognition. Pipelines can be used in many ways, starting from the most general one 'Pipeline()', encapsulating all the pipelines, to task-specific pipelines where we define the task for which we need the pipeline object to be created, e.g., pipeline("text-classification"). We can also specify the model which we want to access with the pipeline, e.g., pipeline(model="roberta-large-mnli"). Once the pipeline is created, we can call it for a single query, e.g., pipe("PWC has approved the loyalty of the firm.") or we can batch the process on a complete dataset. The pipeline object will work like a recursive loop, taking individual queries from the dataset and passing them through the pipeline to the model for that specific task.

We also need to pass a number of other parameters while creating the pipeline object. This includes the mode, tokenizer, boolean full-text, the task, stopping criteria, temperature, max_new_tokens, repetition_penalty. Most of their parameters were already defined in the prerequisite step. Temperature defines the level of freedom a model can have while generating text. A temperature value closer to 0 allows the model to generate answers based on its pre-trained learning, while a value closer to 1 directs the model to be more precise and relevant. Temperature value can help the application avoid Hallucinations. The repetition penalty penalizes the model for generating repeating text. A pipeline is created in Cell#7 in the shared IPython notebook, whereas a similar Hugging Face pipeline is created in Langchain in Cell#7.

A.3 Document Loading

Data is ingested using a document loader. There are a number of document loaders depending on the type of source from which we are loading the data. Langchain supports data ingestion from various sources, including web URLs, text, CSV, docx, GitHub, PDF, and JSON files. It also supports loading multiple documents from a repository. Once the documents are loaded, plain text is extracted from them, which can be seen in Cell#12.

⁶<https://ai.meta.com/resources/models-and-libraries/llama-downloads/>

A.4 Splitting Document into Chunks

Every model has context limitations, also called token limitations. It is the maximum number of tokens that a model can process in a single interaction. LLaMA has a token limit of 4,096, while GPT 4 has increased it to 32,000 tokens. In order to process the file by these token-constrained models, the documents must be split into chunks. Each chunk has a fixed number of tokens while overlapping some with the previous and next token. This way, every physical document results in many logical documents supplemented by metadata. Langchain supports a number of text splitters, including 'RecursiveCharacterTextSplitter,' as explained in Cell#13.

A.5 Creating Embedding and Storing in Vector Storage

We need to store this information in a vector store/database. Vector database stores the data as vector arrays. A vector array is something that AI will understand, and it will assist the AI in finding similar documents when we are looking for them. We must run an embedding algorithm to convert the chunk documents into a vector array. An algorithm converts text into a format our model will understand. We have a number of options to choose from for using the embedding function, e.g., 'sentence-transformers/all-mpnet-base-v2', used in Cell#14. The vector arrays, along with the metadata, will now be stored in the vector storage. There are a number of choices for vector storage, including FAISS (Facebook AI Similarity Search), Chroma, In-Memory Vector, Pinecone, Vectora, and more. Each has strengths and weaknesses and performs well in their domains. Similar embeddings/documents will be stored in clusters so that similar document extraction will become fast in case of similarity search. It will fetch all the documents similar to a user query embedding, normally, the top four documents. These results will be taken as context for generating text by the LLM to respond to the user. We have used FAISS, see Cell#14, which allows developers to search for embeddings of similar documents quickly.

A.6 User Queries and Semantic Search

In the final step, a chain is initialized, e.g., ConversationalRetrievalChain, RetrievalQAChain, SQLiteDatabaseChain, see Cell#15.

B List of Documents Used

Following is the list of documents used with code in Google Colab:

B.1 Energy for a Better Future

Annual Report and Accounts for the Year Ended December 31, 2019, Royal Dutch Shell PLC
FileName: *shell_annual_report_2019.pdf*

B.2 Powering Progress

Annual Report and Accounts for the Year Ended December 31, 2020, Royal Dutch Shell PLC
FileName: *shell-annual-report-2020.pdf*

B.3 Powering Progress

Annual Report and Accounts for the Year Ended December 31, 2021, Royal Dutch Shell PLC
FileName: *shell-annual-report-2021.pdf*

B.4 Powering Progress

Shell PLC Annual Report and Accounts for the Year Ended December 31, 2022
FileName: *shell-annual-report-2022.pdf*

B.5 Our Non-financial Scorecard

PwC UK Annual Report 2022
FileName: *pwc-uk-non-financial-performance-scorecard-2022.pdf*

B.6 PricewaterhouseCoopers LLP

Member's report and financial statements for the financial year ended 30 June 2022
FileName: *pwc-uk-financial-statements-2022.pdf*