Github

https://github.com/GoogleCloudPlatform/generative-ai

This document is an help in understanding the Gen Ai Github repository proposed by Google.
It is not an official document, it is my own work of analysing the repo's content, and the notebooks subject of matter.

It should help a new comer to navigate throught the proposed files, and find a way to cherry-pick where to begin.

The last page propose a summary of each notebook, to find out which speaks about what subject.

I hope you can find it useful !

Romain Jouin / romain.jouin@gmail.com

Repo's structure

```
language
  |-intro_palm_api.ipynb
  |-intro_prompt_design.ipynb
  |-examples
    |-prompt-design
        |-text_summarization.ipynb
        |-ideation.ipynb
        |-question_answering.ipynb
        |-text_classification.ipynb
        |-text_extraction.ipynb
    |-langchain-intro
        |-intro_langchain_palm_api.ipynb
        |-document-summarization
            |-summarization_large_documents_langchain.ipynb
            |-summarization_large_documents.ipynb
        |-document-qa (bots)
            |-question_answering_documents_langchain_matching_engine.ipynb
            |-question_answering_large_documents_langchain.ipynb
            |-question_answering_large_documents.ipynb
            |-utils
                |-matching_engine.py
                |-matching_engine_utils.py
        |-reference-architectures
            |-grocerybot_assistant.ipynb
            |-product_description_generator_image.ipynb
            |-product_description_generator_attributes_to_text.ipynb
    |-tuning
        |-getting_started_tuning.ipynb
```

1)
PALM API
& Prompt design

language

|-intro_palm_api.ipynb

- *The Vertex AI PaLM API enables you to test, customize, and deploy instances of Google's **large language models (LLM)** called as **PaLM**, so that you can leverage the capabilities of PaLM in your applications.*
- *Understand various nuances of generative model **parameters** like **temperature**, **top_k, top_p**, and how each parameter affects the results.*
- *PaLM API currently supports three models:*
  - ***text-bison**@001 : Fine-tuned to follow **natural language instructions** and is suitable for a variety of language tasks.*
  - ***chat-bison**@001 : Fine-tuned for **multi-turn conversation** use cases like building a chatbot.*
  - ***textembedding-gecko**@001 : Returns model **embeddings** for text inputs.*
- *Advanced **Chat model** with the SDK*

language

|-intro_palm_api.ipynb

|-intro_prompt_design.ipynb

- Dealing with **hallucinations** is a fundamental challenge of LLMs and an ongoing research area, so it is important to be cognizant that LLMs may seem to give you confident, correct-sounding statements that are in fact incorrect.
- **Best practices** for prompt engineering:
  - **Be concise**
  - Be specific and well-defined
  - Ask one task at a time
  - Turn generative tasks into **classification tasks**
  - Improve response quality by **including examples** (n-shot prompt)

language
|-intro_palm_api.ipynb : *large language models (LLM), PaLM, parameters, temperature, top_k, top_p, text-bison, chat-bison, textembedding-gecko, Chat model*
|-intro_prompt_design.ipynb : hallucinations, Best practices : concise, classification tasks, examples

2) Prompt design

language
|-intro_palm_api.ipynb
|-intro_prompt_design.ipynb
|-examples
|-**prompt-design**
|-text_summarization.ipynb

- There are two main text **summarization types**:
  - **Extractive** : **selecting** critical **sentences** from the original text and combining them to form a summary.
  - **Abstractive** : **generating** new **sentences** representing the original text's main points
- **Dialogue summarization** involves condensing a conversation into a shorter format so that you don't need to read the whole discussion but can leverage a summary.
- Evaluation : **ROUGE** (Recall-Oriented Understudy for Gisting Evaluation) are measures to automatically determine the quality of a summary by comparing it to other (ideal) summaries created by humans.

language
|-intro_palm_api.ipynb
|-intro_prompt_design.ipynb
|-examples
|-**prompt-design**
|-text_summarization.ipynb
|-ideation.ipynb

- Creating **reading comprehension questions** - You can use the PaLM API to generate some example questions to test a person's understanding of a provided passage of text.
- **Interview question** generation
- **Impersonation**

language
|-intro_palm_api.ipynb
|-intro_prompt_design.ipynb
|-examples
|-**prompt-design**
|-text_summarization.ipynb
|-ideation.ipynb
|-question_answering.ipynb

- When creating a **question-answering prompt**, it is essential to be specific and provide as much context as possible. It helps the model understand the intent behind the question and generate a relevant response. For example, if you want to ask: "What is the capital of France?", then a good prompt could be:
  - "Please tell me the name of the city that serves as the capital of France."
- **Open domain** questions: All questions whose answers are available online already (n-shot prompting)
- **Closed domain** questions: All questions whose answers are available online already. You can pass that "private" knowledge as context to the model.
  - Providing custom knowledge as **context**
  - **Instruction-tune** the outputs
  - **Few-shot** prompting
- **Extractive Question-Answering** : The model is given a question and a passage of text, and is asked to find the answer to the question within the passage. The answer is typically a phrase or sentence.

language

|-intro_palm_api.ipynb

|-intro_prompt_design.ipynb

|-examples

|-**prompt-design**

|-text_summarization.ipynb

|-ideation.ipynb

|-question_answering.ipynb

|-text_classification.ipynb

- Build **a text classification solution** assigning one or more categories to a given piece of text.
- n-shot prompting for text classification tasks :
  - **“Classify” + task**
    - **Labels** (dogs, cats)
    - **Topics** (business / entertainment / health / sports / technology)
    - **Intent** recognition ("finding information", "making a reservation", or "placing an order")
    - **Language** identification, **Toxicity** detection, **Emotion** detection…

language
|-intro_palm_api.ipynb
|-intro_prompt_design.ipynb
|-examples
|-**prompt-design**
|-text_summarization.ipynb
|-ideation.ipynb
|-question_answering.ipynb
|-text_classification.ipynb
|-text_extraction.ipynb

- One common purpose is to **convert documents into a machine-readable format**. Useful for storing documents in a database or for processing documents with software.
- Another common purpose is to **extract information from documents**. This can be useful for finding specific information in a document or for **summarizing** the content of a document.
- Extract information that is not explicitly stated or fill in the gaps in text that is missing information
- The answers from LLMs can also be further improved through methods like few-shot prompting.
- Examples :
  - Extract technical specifications **from Json** (pixel phone)
  - Answering a customer's question based on **provided documentation** (WiFi troubleshooting)
  - Respond to inquiries in character / **tell the model about personality traits** of characters (
  - **Converting a list to JSON** format
  - **Organizing the results** of a text extraction

|-examples
  |-prompt-design
    |- text_summarization.ipynb : summarization types: Extractive (selecting sentences), Abstractive (generating sentences), Dialogue summarization, ROUGE
    |- ideation.ipynb : creating comprehension questions, Interview question, Impersonation
    |- question_answering.ipynb : question-answering prompt, Open domain Vs Closed domain, context, Instruction-tune output, Few-shot, Extractive Question-Answering
    |- text_classification.ipynb : a text classification solution - “Classify” + Labels / Topics / Intent / Language / Toxicity / Emotion
    |- text_extraction.ipynb : convert documents into a machine-readable format, extract information from documents, summarizing - from Json / from provided documentation /
                                tell the model about personality traits of characters / Converting a list to JSON / Organizing the results

3) Lang chain intro

language

|-intro_palm_api.ipynb

|-intro_prompt_design.ipynb

|-examples

|-**prompt-design**

|-text_summarization.ipynb

|-ideation.ipynb

|-question_answering.ipynb

|-text_classification.ipynb

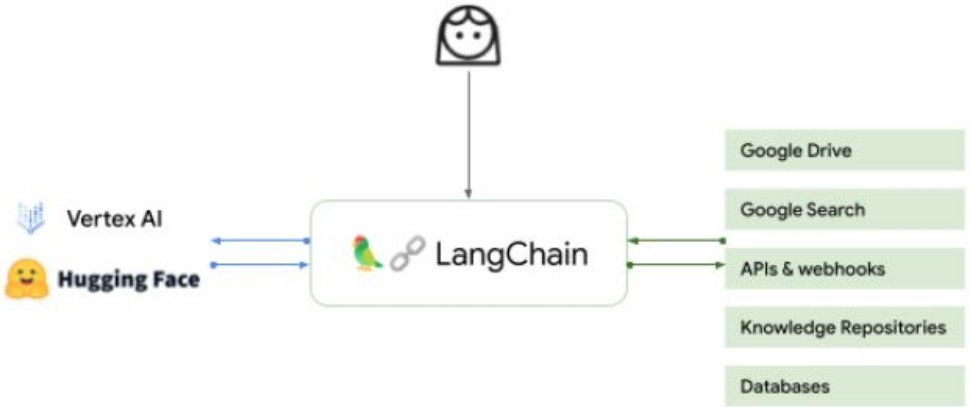
|-text_extraction.ipynb

|-langchain-intro

|-intro_langchain_palm_api.ipynb

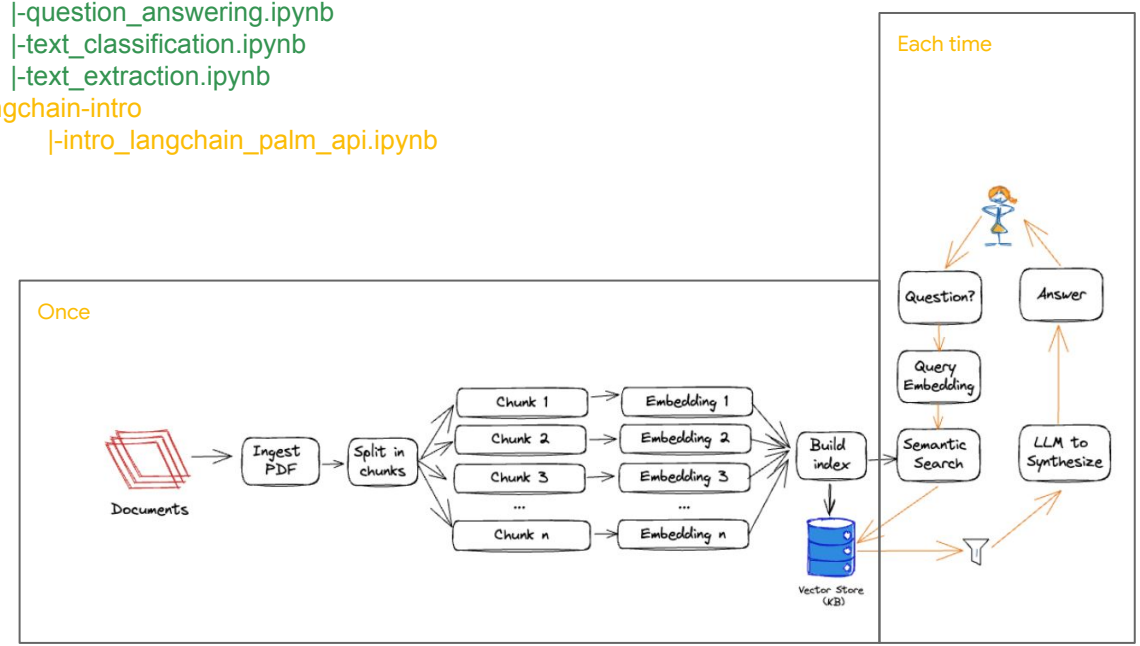
- LangChain is a framework for developing applications powered by large language models (LLMs).
- It helps do this in two ways:
  - **Integration** : **Bring external data**, such as your files, other applications, and API data, **to LLMs**
  - **Agents** : Allows LLMs to interact with its environment via **decision making** and use LLMs to help decide which action to take next
- This **makes models data-aware and agentic**, meaning they can understand, reason, and use data to take action in a meaningful way
  - Convert **natural language to SQL**, executing the SQL on database, analyze and present the results
  - Calling an external **webhook or API** based on the user query
  - Synthesize outputs from multiple models, or **chain the models** in a specific order
  - LangChain is a kind of **glue code**
  - **Memory** provides a construct for storing and retrieving messages during a conversation
  - **Indexes** help LLMs interact with documents by providing a way to structure them. LangChain provides **Document Loaders**, **Text Splitters**, **Vector Stores** to, and **Retrievers** to fetch relevant documents.
- **Vertex AI PaLM foundational models — Text, Chat, and Embeddings — are officially integrated with the LangChain Python SDK**, making it convenient to build applications on top of Vertex AI PaLM models.

language
- |-intro_palm_api.ipynb
- |-intro_prompt_design.ipynb
- |-examples
  - |-**prompt-design**
    - |-text_summarization.ipynb
    - |-ideation.ipynb
    - |-question_answering.ipynb
    - |-text_classification.ipynb
    - |-text_extraction.ipynb
  - |-langchain-intro
    - |-intro_langchain_palm_api.ipynb



language
- |-intro_palm_api.ipynb
- |-intro_prompt_design.ipynb
- |-examples
  - |-**prompt-design**
    - |-text_summarization.ipynb
    - |-ideation.ipynb
    - |-question_answering.ipynb
    - |-text_classification.ipynb
    - |-text_extraction.ipynb
  - |-langchain-intro
    - |-intro_langchain_palm_api.ipynb

language
- |-intro_palm_api.ipynb
- |-intro_prompt_design.ipynb
- |-examples
  - |-**prompt-design**
    - |-text_summarization.ipynb
    - |-ideation.ipynb
    - |-question_answering.ipynb
    - |-text_classification.ipynb
    - |-text_extraction.ipynb
  - |-langchain-intro
    - |-intro_langchain_palm_api.ipynb



|-langchain-intro
  |- intro_langchain_palm_api.ipynb : LangChain as glue code for Integration : Bring external data to LLMs + Agents = makes models data-aware and agentic (webhook or API) - chain the models (Memory + Indexes)

4) Document summarization

language

|-intro_palm_api.ipynb

|-intro_prompt_design.ipynb

|-examples

|-**prompt-design**

|-text_summarization.ipynb

|-ideation.ipynb

|-question_answering.ipynb

|-text_classification.ipynb

|-text_extraction.ipynb

|-langchain-intro

|-intro_langchain_palm_api.ipynb

|-document-summarization

|-summarization_large_documents_langchain.ipynb

- **Summarizing large documents** can be challenging. To create summaries, you need to apply summarization strategies to your indexed documents :
  - **Stuffing method** : "stuffs" text into the prompt as context - If the document is too long, the stuffing method will not work
  - **MapReduce method** : multi-stage summarization. It is a technique for summarizing large pieces of text by first summarizing smaller chunks of text and then combining those summaries into a single summary.
  - **Refine method** : first running an initial prompt on a small chunk of data, generating some output. Then, for each subsequent document, the output from the previous document is passed in along with the new document, and the LLM is asked to refine the output based on the new document.

language

|-intro_palm_api.ipynb

|-intro_prompt_design.ipynb

|-examples

|-**prompt-design**

|-text_summarization.ipynb

|-ideation.ipynb

|-question_answering.ipynb

|-text_classification.ipynb

|-text_extraction.ipynb

|-langchain-intro

|-intro_langchain_palm_api.ipynb

|-document-summarization

|-summarization_large_documents_langchain.ipynb

|-summarization_large_documents.ipynb

- **Summarizing large documents** can be challenging. To create summaries, you need to apply summarization strategies to your indexed documents :
  - **Stuffing method** : "stuffs" text into the prompt as context - If the document is too long, the stuffing method will not work
    - Adding **rate limit** to model calls
  - **MapReduce method** : multi-stage summarization. first summarizing smaller chunks of text and then combining those summaries into a single summary.
  - **MapReduce with Overlapping Chunks** : a few pages will be summarized together, rather than each page being summarized separately. This helps to preserve more context or information between chunks, which can improve the accuracy of the results.
  - **MapReduce with Rolling Summary(Refine)** : first running an initial prompt on a small chunk of data, generating some output. Then, for each subsequent document, the output from the previous document is passed in along with the new document, and the LLM is asked to refine the output based on the new document.

```
|- document-summarization
        |-summarization_large_documents_langchain.ipynb : Summarizing large documents : Stuffing method / MapReduce method / Refine method
        |-summarization_large_documents.ipynb : Summarizing large documents : MapReduce with Overlapping Chunks / MapReduce with Rolling Summary
```

5) Document question and answer

language

|-examples

|-**prompt-design**

|-langchain-intro

|-intro_langchain_palm_api.ipynb

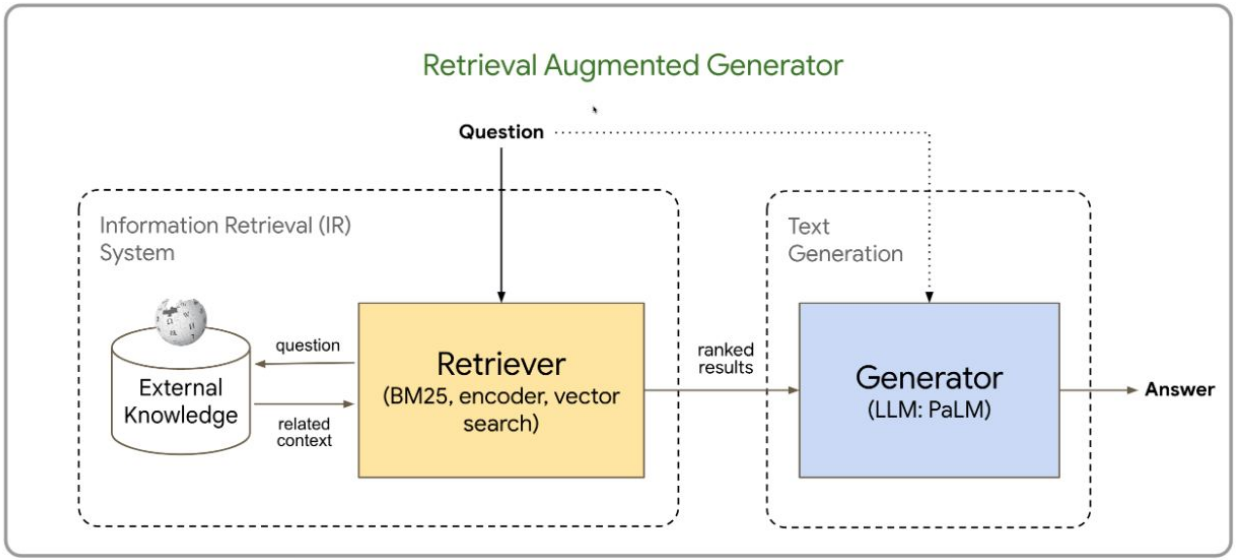
|-document-summarization

|-document-qa (bots)

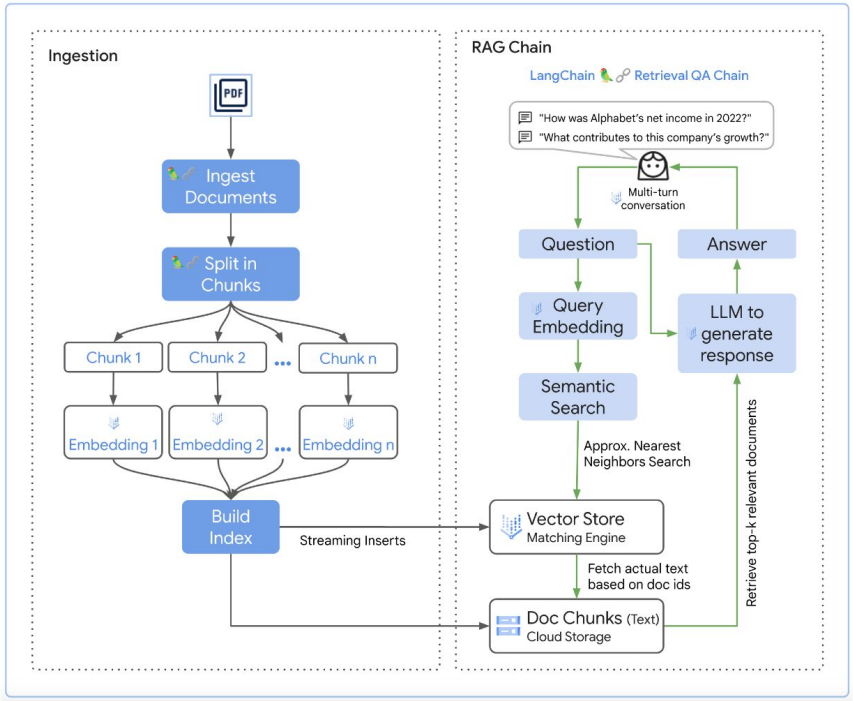
|-question_answering_documents_langchain_matching_engine.ipynb

- Augmenting LLM's knowledge with **external data sources** such as documents, websites.
- **Augment the prompt** with relevant data from knowledge base: **Information Retrieval (IR) mechanism**
- This approach is called **Retrieval Augmented Generation (RAG)**, also known as **Generative QA** in the context of the QA task. Two main components in RAG based architecture:
  a. **Retriever** : retrieve relevant snippets from documents (based on the user's query) -> passed as "**context**" to the Generator.
  b. **Generator** : The context is passed to an LLM to generate a well **grounded response**
- First we need to ingest knowledge in a vector store
  a. Generate embeddings for each chunked document
- Second, prompt the model :
  a. The question is embedded in the same vector space, and a k-nearest neighbor match is passed to the LLM

language
|-examples
|-**prompt-design**
|-langchain-intro
|-intro_langchain_palm_api.ipynb
|-document-summarization
|-document-qa (bots)
|-question_answering_documents_langchain_matching_engine.ipynb



language
|-examples
|-**prompt-design**
|-langchain-intro
|-intro_langchain_palm_api.ipynb
|-document-summarization
|-document-qa (bots)
|-question_answering_documents_langchain_matching_engine.ipynb



language

|-examples

|-**prompt-design**

|-langchain-intro

|-intro_langchain_palm_api.ipynb

|-document-summarization

|-document-qa (bots)

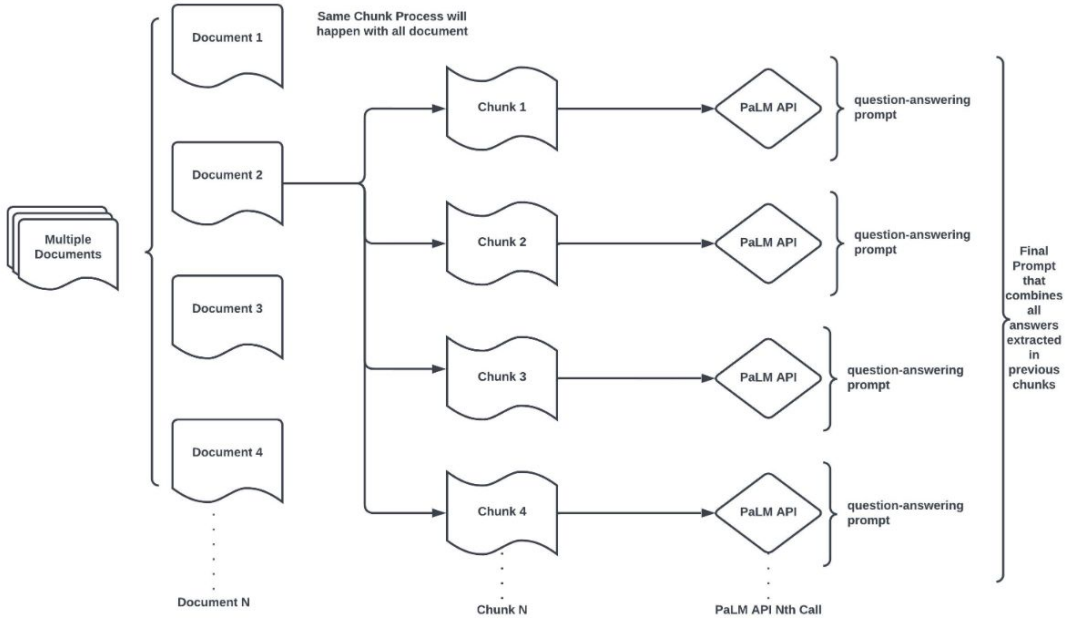
|-question_answering_documents_langchain_matching_engine.ipynb

|-question_answering_large_documents_langchain.ipynb

- Answer a wide range of questions about large document base
- The challenge with building a Q&A system over large documents is that Large Language Models, **LLMs in short, have token limits that restrict how much context you can provide.** There are several methods to provide the context :
  - **Stuffing:** Push the whole document content as a context.
  - **Map-Reduce**: Split documents into smaller chunks and process them in parallel.
  - **Refine**: Run an initial prompt on a small chunk, generate an output and for each subsequent document, refine the output based on both output and new document.
  - **Map-Reduce with Similarity search :** you create embeddings of smaller chunks and use vector similarity search to find relevant context. This is the most efficient method, but it can be the most complex to implement.
- These methods are explained in question_answering_large_documents.ipynb

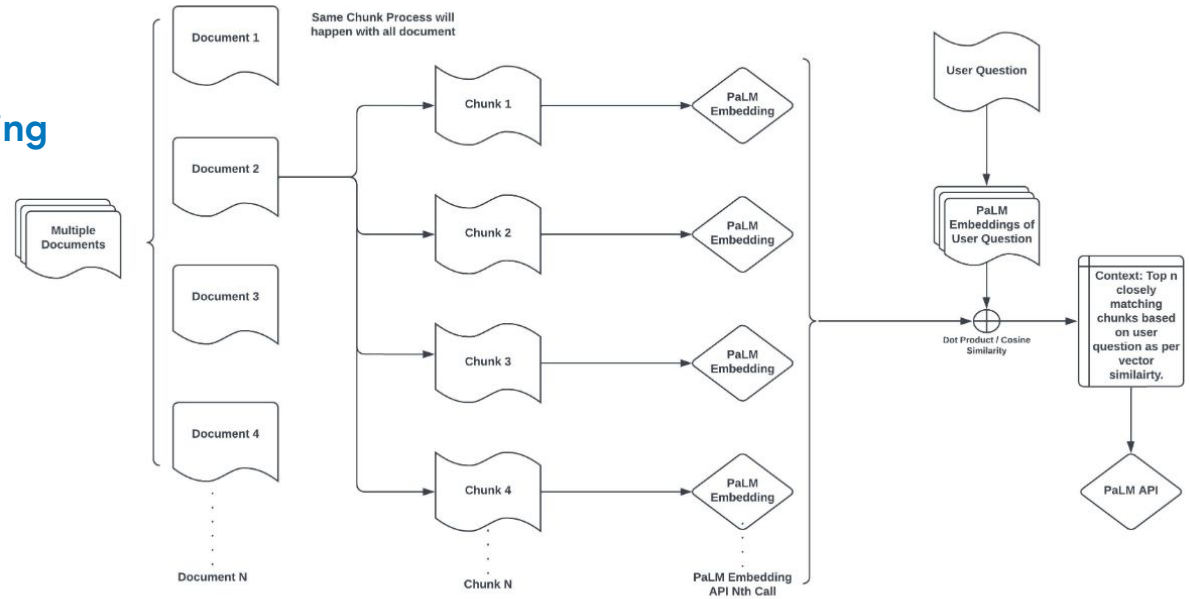
language
|-examples
|-**prompt-design**
|-langchain-intro
|-intro_langchain_palm_api.ipynb
|-document-summarization
|-document-qa (bots)
|-question_answering_documents_langchain_matching_engine.ipynb
|-question_answering_large_documents_langchain.ipynb

Map reduce



language
|-examples
|-**prompt-design**
|-langchain-intro
|-intro_langchain_palm_api.ipynb
|-document-summarization
|-document-qa (bots)
|-question_answering_documents_langchain_matching_engine.ipynb
|-question_answering_large_documents_langchain.ipynb

Map
reduce
with
embedding



|- document-qa (bots)

|-question_answering_documents_langchain_matching_engine.ipynb : external data sources / Augment the prompt with Information Retrieval (IR) mechanism / Retrieval / Augmented Generation (RAG) / Retriever & Generator = grounded response

|-question_answering_large_documents_langchain.ipynb : LLMs in short, have token limits that restrict how much context you can provide / solutions = Stuffing / Map-Reduce / Refine / Map-Reduce with Similarity search

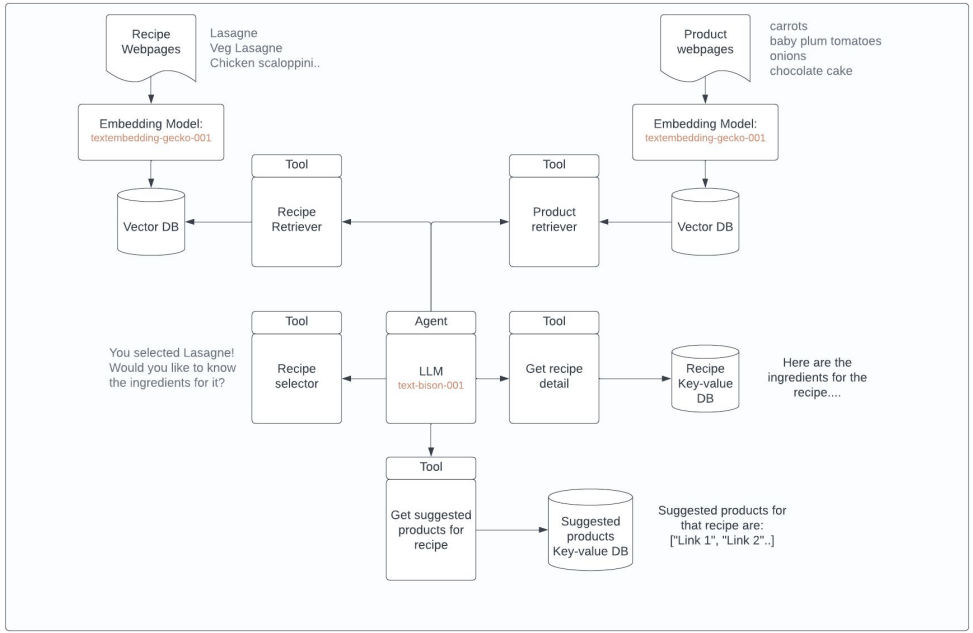
|-question_answering_large_documents.ipynb

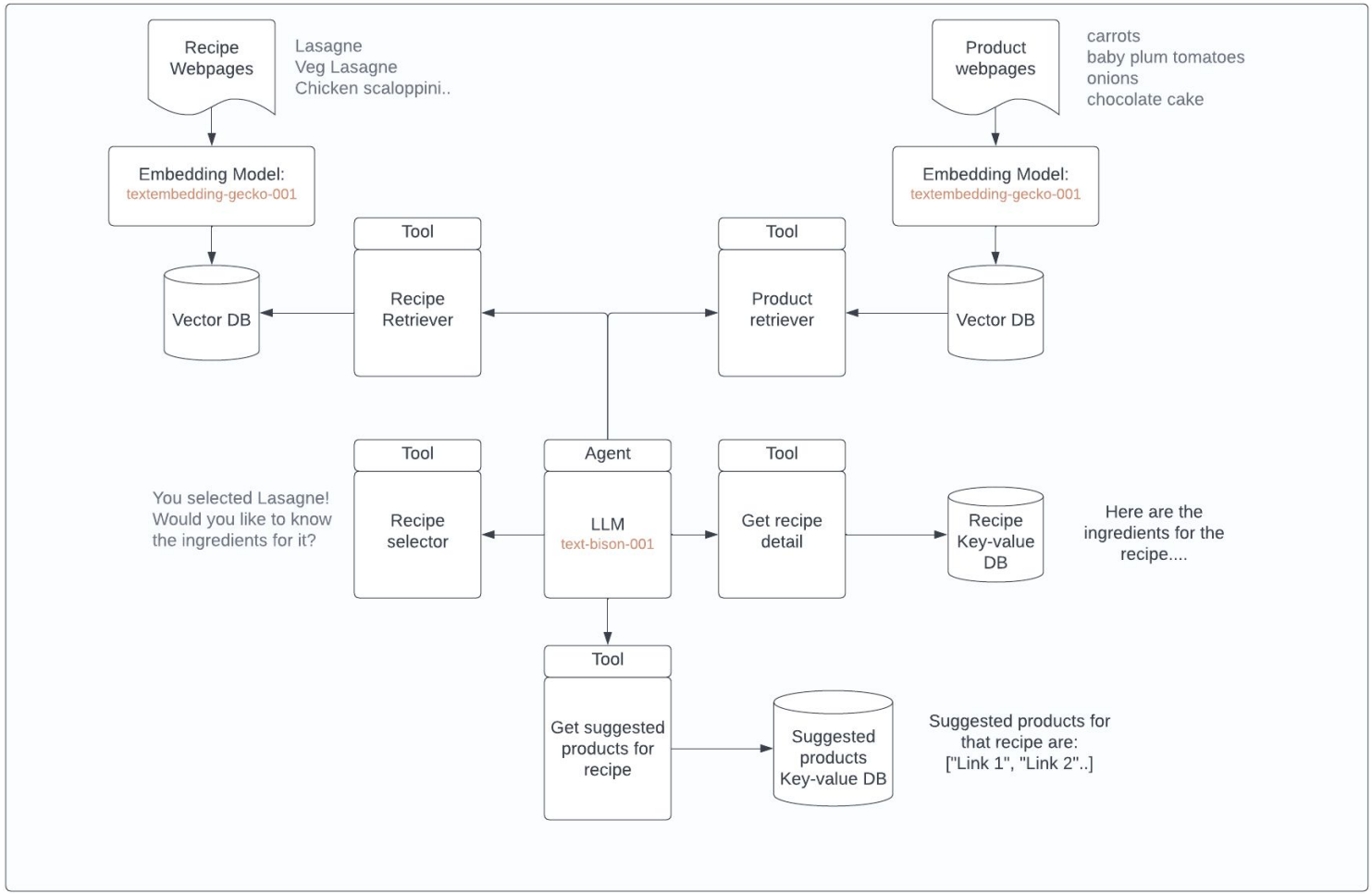
6) Reference architecture

language
|-examples
|-**prompt-design**
|-langchain-intro
|-intro_langchain_palm_api.ipynb
|-document-summarization
|-document-qa (bots)
|-reference-architectures
|-grocerybot_assistant.ipynb

- Create a **conversational bot**, capable of assisting a customer in their grocery shopping journey (recipe & ingredients)
- To do so you will create **two retrievers** in Langchain, capable of interacting with the two **vector databases**:
  - One retriever for the product items / another retriever for the recipe items.
- **As a one off process** :
  - every product and recipe item will be converted into an embedding and ingested into the relevant vector database.
- At retrieval time :
  - the query (e.g. lasagne) will be converted into an embedding, and a **vector similarity search** will be performed to find the closest items to the query (e.g. lasagne al forno, vegetarian lasagne).
- Connecting LLMs with **external knowledge sources**, such as databases (Langchain TextLoader / LangChain WebBaseLoader)
  - **Retrieval Augmented Generation (RAG)** : which attempts to **mitigate the problem of hallucination** by **inserting factual information into the prompt** which is sent to the LLM.
  - **Implementing a Reasoning + Acting (ReAct) - like Agent** in Langchain, capable of taking decisions and decide when to query these databases.
    - **An Agent** has access to a suite of tools, which you can think of as Python functions that can potentially do anything you equip it with. What makes the Agent setup unique is its **ability to autonomously decide which tool to call** and in which order, **based on the user input**.
      - We need to **create the tools the agent will use**. For each tool, it's critical to **provide a good description of what the tool does**, as it will be used by the agent to perform actions (uses the `tool` decorator approach)
      - You will provide to the agent a memory so to allow a conversation.
      - what if the user uses the agent to perform things they are not allowed to ? => **Set Agent's guardrails**

language
|-examples
|-**prompt-design**
|-langchain-intro
|-intro_langchain_palm_api.ipynb
|-document-summarization
|-document-qa (bots)
|-reference-architectures
|-grocerybot_assistant.ipynb




Recipe Webpages
Lasagne
Veg Lasagne
Chicken scaloppini..
Product webpages
carrots
baby plum tomatoes
onions
chocolate cake
Embedding Model:
textembedding-gecko-001
Embedding Model:
textembedding-gecko-001
Tool
Recipe Retriever
Tool
Product retriever
Vector DB
Vector DB
Tool
Recipe selector
Agent
LLM
text-bison-001
Tool
Get recipe detail
Recipe Key-value DB
You selected Lasagne! Would you like to know the ingredients for it?
Here are the ingredients for the recipe....
Tool
Get suggested products for recipe
Suggested products Key-value DB
Suggested products for that recipe are:
["Link 1", "Link 2"..]

language

|-examples

|-**prompt-design**

|-langchain-intro

|-intro_langchain_palm_api.ipynb

|-document-summarization

|-document-qa (bots)

|-reference-architectures

|-grocerybot_assistant.ipynb

|-product_description_generator_image.ipynb

- **Product descriptions from images**. In this notebook, you will use Fashion Image Dataset to create product descriptions for the clothing images.
- Using Huggingface's blip-image-captioning model
  - Upload the model to **Model Registry**.
  - Deploy the model on Endpoint.
  - Run online predictions for image captioning.
  - Run online predictions for PaLM model with the image captions to produce product descriptions.

language

|-examples

|-**prompt-design**

|-langchain-intro

|-intro_langchain_palm_api.ipynb

|-document-summarization

|-document-qa (bots)

|-reference-architectures

|-grocerybot_assistant.ipynb

|-product_description_generator_image.ipynb

|-product_description_generator_attributes_to_text.ipynb

- Automatic generation of informative, **SEO-optimized**, and potentially creative **product descriptions and titles**, based on product attributes or specifications (often provided by a supplier)
- Use **retrieval-augmented generation (RAG)** : which attempts to **mitigate the problem of hallucination** by **inserting factual information into the prompt** which is sent to the LLM through k-NN (k-nearest neighbor) embedding search.
  - N-shot prompting
  - Prompt-engineered LLM zero-shot prompts to check the safety, veracity, and quality of generated product descriptions, and generate a reasoning for its evaluation
  - Evaluating generated prompts using n-gram overlap metrics like BLEU and ROUGE
  - Prompt templates / LLM chains / sequential LLM chains / k-NN retrievers / custom LLM classes
- Methods :
  - **Zero-shot** description generation, validation & evaluation
  - **Few-shot** description generation using dynamic k-nearest neighbours
  - **Fine-tuned zero-shot** description generation validation & evaluation
    - fine tuning of the model on 500 randomly sampled (prompt, description) pairs from the training dataset (**needs TPU**)
  - **Few-shot description** generation validation & evaluation **using fine-tuned models** (**needs TPU**)

```
|- reference-architectures
        |-grocerybot_assistant.ipynb : conversational bot / external knowledge sources / Retrieval Augmented Generation (RAG = inserting factual information into the
                                       prompt) / Reasoning + Acting (ReAct) - like Agent (ability to autonomously decide which tool to call - provide a good description of what the tool ) +
                                       Set Agent's guardrails
         |-product_description_generator_image.ipynb : Product descriptions from images / Huggingface's blip model / Model Registry
         |-product_description_generator_attributes_to_text.ipynb : SEO-optimized product descriptions and titles
```

7) Model Tuning

language
|-examples
|-**prompt-design**
|-reference-architectures
|-langchain-intro
|-document-qa
|-document-summarization
|-tuning
| |-getting_started_tuning.ipynb

- Tuning allows you to **customize a foundation model for a more specific task** or knowledge domains, based on examples.
- Important: Tuning the text-bison@001 model **uses the tpu**-v3-8 training resource
- Tune a foundation model by creating a **pipeline job**
- Your model tuning dataset must be in a **JSONL format** (json with newline as separator) where each line contains a single example.
- Recommended Tuning Configurations
  - Make sure that your **train dataset size is 100+**
  - **Training steps** [100-500]
- Your model will be available on **Vertex AI Model Registry**

|-tuning
  |-getting_started_tuning.ipynb : customize a foundation model for a more specific task / uses the tpu / pipeline job / JSONL format / train dataset size is 100+ / Training steps / Vertex AI Model Registry

8) Summary

language

|-intro_prompt_design.ipynb : hallucinations, Best practices : concise, classification tasks, examples

|-intro_palm_api.ipynb : *large language models (LLM), PaLM, parameters, temperature, top_k, top_p, text-bison, chat-bison, textembedding-gecko, Chat model*

|-examples

|-prompt-design

|- text_summarization.ipynb : summarization types: Extractive (selecting sentences), Abstractive (generating sentences), Dialogue summarization, ROUGE

|- ideation.ipynb : creating comprehension questions, Interview question, Impersonation

|- question_answering.ipynb : question-answering prompt, Open domain Vs Closed domain, context, Instruction-tune output, Few-shot, Extractive Question-Answering

|- text_classification.ipynb : a text classification solution - “Classify” + Labels / Topics / Intent / Language / Toxicity / Emotion

|- text_extraction.ipynb : convert documents into a machine-readable format, extract information from documents, summarizing - from Json / from provided documentation / tell the model about personality traits of characters / Converting a list to JSON / Organizing the results

|-langchain-intro

|- intro_langchain_palm_api.ipynb : LangChain as glue code for Integration : Bring external data to LLMs + Agents = makes models data-aware and agentic (webhook or API) - chain the models (Memory + Indexes)

|- document-summarization

|-summarization_large_documents_langchain.ipynb : Summarizing large documents : Stuffing method / MapReduce method / Refine method

|-summarization_large_documents.ipynb : Summarizing large documents : MapReduce with Overlapping Chunks / MapReduce with Rolling Summary

|- document-qa (bots)

|-question_answering_documents_langchain_matching_engine.ipynb : external data sources / Augment the prompt with Information Retrieval (IR) mechanism / Retrieval / Augmented Generation (RAG) / Retriever & Generator = grounded response

|-question_answering_large_documents_langchain.ipynb : LLMs in short, have token limits that restrict how much context you can provide / solutions = Stuffing / Map-Reduce / Refine / Map-Reduce with Similarity search

|-question_answering_large_documents.ipynb

|- reference-architectures

|-grocerybot_assistant.ipynb : conversational bot / external knowledge sources / Retrieval Augmented Generation (RAG = inserting factual information into the prompt) / Reasoning + Acting (ReAct) - like Agent (ability to autonomously decide which tool to call - provide a good description of what the tool) + Set Agent’s guardrails

|-product_description_generator_image.ipynb : Product descriptions from images / Huggingface's blip model / Model Registry

|-product_description_generator_attributes_to_text.ipynb : SEO-optimized product descriptions and titles

|-tuning

|-getting_started_tuning.ipynb : customize a foundation model for a more specific task / uses the tpu / pipeline job / JSONL format / train dataset size is 100+ / Training steps / Vertex AI Model Registry