EE6405 NLP PROJECT PRESENTATION

Group A6

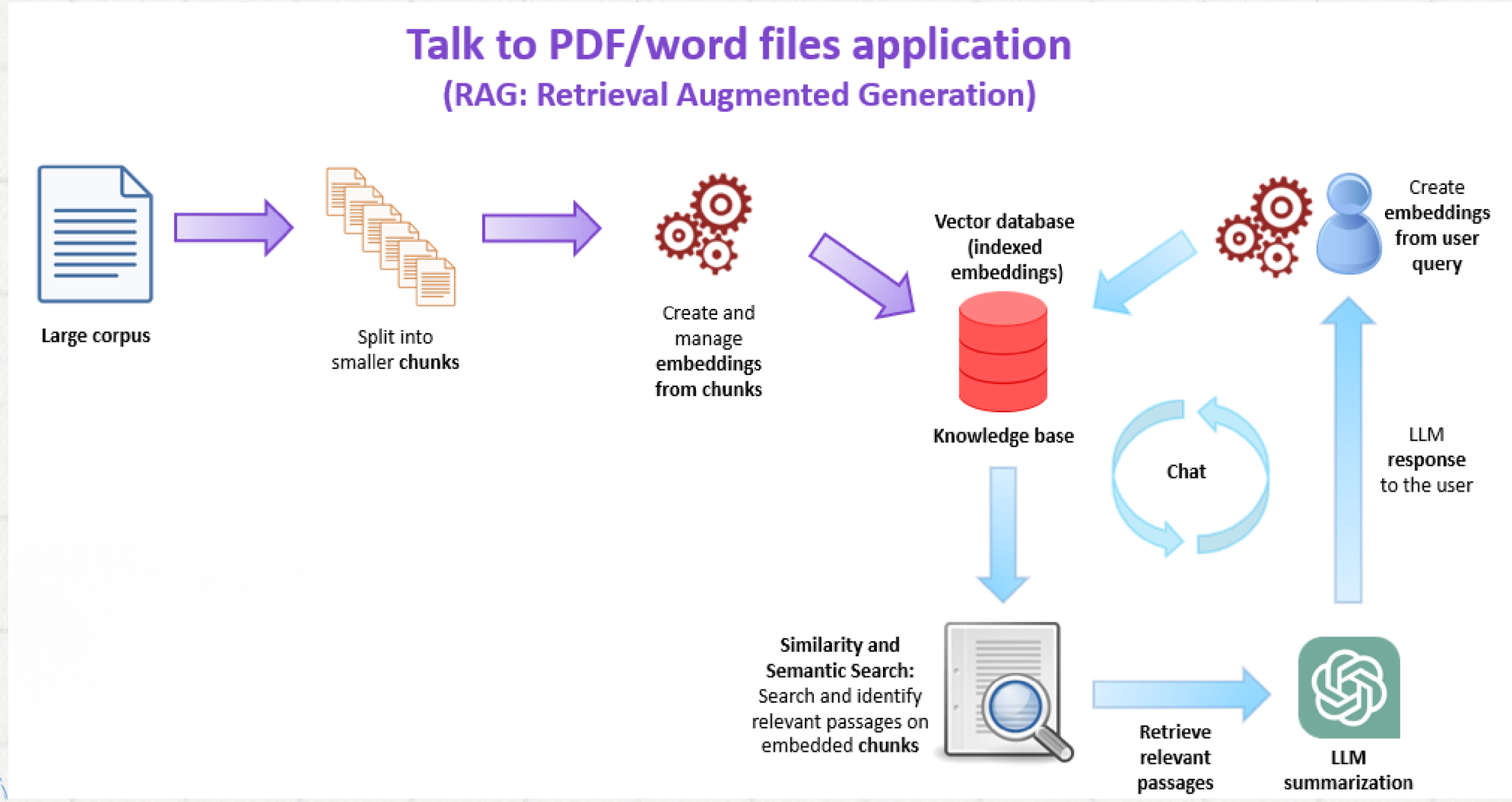
# INTERACTIVE Q&A BOT EMPOWERED BY LLM(MISTRAL-7B)

**Members**: Guo Beiting, Liu Chang, Yuan Weiyun,  Sun Chuhang, Pan Jiahang, Murugan Paramesh

# INTRODUCTION

- Develop a chatbot using Langchain that can talk to PDF or any other pdf files to answer questions.
- Utilizing the Natural Language Processing (NLP) **resources provided by EE6405 course** to obtain the accurate response from the chatbot.
- Flexible and customizable **RAG pipeline** (Retrieval Augmented Generation).
- **FAISS** vector store to save the text embeddings generated by Sentence Transformers.
- Employ a retrieval chain to extract relevant passages from the embedded text and provide a concise overview of the passages.
- "**Gradio"** as chat UI.

# PROJECT EXPLANATION



# Context-free Grammar(CFG)

```
class CFG:
    # LLMs
    model_name = 'mistral-7B' # wizardlm, llama2-7b-chat, llama2-13b-chat, mistral-7B
    temperature = 0
    top_p = 0.95
    repetition_penalty = 1.5

    # splitting
    split_chunk_size = 200
    split_overlap = 0

    # embeddings
    embeddings_model_repo = 'sentence-transformers/all-MiniLM-L6-v2'

    # similar passages
    k = 4

    # paths
    PDFs_path = 'sample_data/'
    Embeddings_path =  'faiss-hp-sentence-transformers'
    Output_folder = './vectordb'
```

**Model name:** specifies the name of the selected language model. In this example, the model name is' missing 7B ', but other alternative options are also provided.

**Top-p**: The top-p or nucleus sampling parameter used to generate text, which controls the strategy of selecting the next word from the probability distribution. Here, it is set to 0.95.

**Repetition_penalty:** A repetition penalty parameter used to generate text, which controls the degree of duplicate content in the generated text. Here, it is set to 1.5.

# Model

```
def get_model(model = CFG.model_name):

    print('\nDownloading model: ', model, '\n\n')

    if model == 'wizardlm':
        model_repo = 'TheBloke/wizardLM-7B-HF'

        tokenizer =  .from_pretrained(model_repo)

        bnb_config = BitsAndBytesConfig(
            load_in_4bit = True,
            bnb_8bit_quant_type = "nf4",
            bnb_8bit_compute_dtype = torch.float16,
            bnb_8bit_use_double_quant = True,
        )

        model = AutoModelForCausalLM.from_pretrained(
            model_repo,
            quantization_config = bnb_config,
            device_map = 'auto',
            low_cpu_mem_usage = True
        )

        max_len = 1024
```

Set the model-repo variable to specify the repository for the pre trained model.

Load the pre trained model's tokenizer using the AutoTokenizer. free_pretrained() method.

Use BitsAndBytesConfig to configure parameters related to bits and bytes for quantization.

Use the AutoModelForCausalLM. free_pretrained() method to load the backbone of the pre trained model.

Set the max_len variable to represent the maximum input sequence length.

# Llama2, Mistral-7B

```
elif model == 'llama2-13b-chat':
    model_repo = 'daryl149/llama-2-13b-chat-hf'

    tokenizer = AutoTokenizer.from_pretrained(model_repo, use_fast=True)

    bnb_config = BitsAndBytesConfig(
        load_in_4bit = True,
        bnb_4bit_quant_type = "nf4",
        bnb_4bit_compute_dtype = torch.float16,
        bnb_4bit_use_double_quant = True,
    )

    model = AutoModelForCausalLM.from_pretrained(
        model_repo,
        quantization_config = bnb_config,
        device_map = 'auto',
        low_cpu_mem_usage = True,
        trust_remote_code = True
    )

    max_len = 2048 # 8192
```

```
elif model == 'mistral-7B':
    model_repo = 'mistralai/Mistral-7B-v0.1'

    tokenizer = AutoTokenizer.from_pretrained(model_repo)

    bnb_config = BitsAndBytesConfig(
        load_in_4bit = True,
        bnb_4bit_quant_type = "nf4",
        bnb_4bit_compute_dtype = torch.float16,
        bnb_4bit_use_double_quant = True,
    )

    model = AutoModelForCausalLM.from_pretrained(
        model_repo,
        quantization_config = bnb_config,
        device_map = 'auto',
        low_cpu_mem_usage = True,
    )

    max_len = 1024

else:
    print("Not implemented model (tokenizer and backbone)")

return tokenizer, model, max_len
```

# Pipeline

```
### hugging face pipeline
pipe = pipeline(
    task = "text-generation",
    model = model,
    tokenizer = tokenizer,
    pad_token_id = tokenizer.eos_token_id,
#     do_sample = True,
    max_length = max_len,
    temperature = CFG.temperature,
    top_p = CFG.top_p,
    repetition_penalty = CFG.repetition_penalty
)

### langchain pipeline
llm = HuggingFacePipeline(pipeline = pipe)
```

This code uses two important components from the Hugging Face library: pipeline and Hugging FacePipeline.

The pipeline function creates a Hugging Face pipeline for executing various NLP tasks.Here, the task type is specified as "text generation".
The model and tokenizer parameters specify the pre trained language model and word segmentation.

The padutoken_id parameter specifies the ID of the padding tag, usually the ID of the sentence end tag, to ensure that the output is a complete sentence.

The max_length parameter specifies the maximum length of generated text.

The temperature, top-p, and repetition_penalty parameters set the temperature, top-p, and repetition penalty parameters during text generation.

# Process Data with LangChain library

Langchain is one of the important libraries used in this project. This library helps the creation of embeddings and the handling of data.

01. Dataloader

02. Splitter

03. Create Embeddings

# Dataloader and Splitter

```
%%time

loader = DirectoryLoader(
    CFG.PDFs_path,
    glob="./*.pdf",
    loader_cls=PyPDFLoader,
    show_progress=True,
    use_multithreading=True
)

documents = loader.load()
```

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = CFG.split_chunk_size,
    chunk_overlap = CFG.split_overlap
)

texts = [text for doc in documents for text in text_splitter.split_documents(doc)]
print(f'We have created {len(texts)} chunks')
```

## 1.Check for existing index

```
### we create the embeddings only if they do not exist yet
if not os.path.exists(CFG.Embeddings_path + '/index.faiss'):
```

## 2.Download embedding model

```
### download embeddings model
embeddings = HuggingFaceInstructEmbeddings(
    model_name = CFG.embeddings_model_repo,
    model_kwargs = {"device": "cuda"},
)
```

## 3.Create Embeddings and database

```
### create embeddings and DB
vectordb = FAISS.from_documents(
    documents = texts,
    embedding = embeddings
)
```

## 4.Persist vector database

```
### persist vector database
vectordb.save_local(f"{CFG.Output_folder}/faiss_index_hp")
  vectordb.save_local(f"{CFG.Embeddings_path}/faiss_index_hp")
```

# Load Vector Database

```
### download embeddings model
embeddings = HuggingFaceInstructEmbeddings(
    model_name = CFG.embeddings_model_repo,
    model_kwargs = {"device": "cuda"}
)

### load vector DB embeddings
vectordb = FAISS.load_local(
    CFG.Embeddings_path, # from input folder
#     CFG.Output_folder + '/faiss_index_hp', # from output folder
    embeddings
)

clear_output()
```

This code uses the Hugging Face Transformers library to download **a pretrained embeddings model** and **load vector database** embeddings stored locally.

Obviously, the embeddings function to load the embeddings must be the same as the one used to create the embeddings.

# Prompt Template

```
prompt_template = """

PROMPT = PromptTemplate(
    template = prompt_template,
    input_variables = ["context", "question"]
)
```

This code defines
**a template**.

# Retriever Chain

```
retriever = vectordb.as_retriever(search_kwargs = {"k": CFG.k, "search_type" :
similarity"})

qa_chain = RetrievalQA.from_chain_type(
    llm = llm,
    chain_type = "stuff", # map_reduce, map_rerank, stuff, refine
    retriever = retriever,
    chain_type_kwargs = {"prompt": PROMPT},
    return_source_documents = True,
    verbose = False
)
```

This code creates **a question answering (QA) system**. It utilizes an embedded vector database and a QA chain to retrieve similar vectors from the provided question and generate answers based on the database.

# Post-process outputs

This Python code automatically wraps the input text according to a specified line width , while preserving the original positions of the newline characters. This processed text becomes more suitable for viewing in a fixed-width display environment without causing line breaks to span across the display boundaries
.

```python
def wrap_text_preserve_newlines(text, width=700):
    # Split the input text into lines based on newline characters
    lines = text.split('\n')

    # Wrap each line individually
    wrapped_lines = [textwrap.fill(line, width=width) for line in lines]

    # Join the wrapped lines back together using newline characters
    wrapped_text = '\n'.join(wrapped_lines)

    return wrapped_text
```

```python
def process_llm_response(llm_response):
    ans = wrap_text_preserve_newlines(llm_response['result'].split("Answer:")[1].strip())
    if "Question:" in ans:
        ans = ans.split("Question:")[0].strip()
    # Remove duplicate text
    sentences = ans.split('. ')
    unique_sentences = []
    for sentence in sentences:
        if sentence not in unique_sentences:
            unique_sentences.append(sentence)
    ans = '. '.join(unique_sentences)
    # sources_used = ' \n'.join(
    #     [
    #         source.metadata['source'].split('/')[-1][:-4]
    #         + ' - page: '
    #         + str(source.metadata['page'])
    #         for source in llm_response['source_documents']
    #     ]
    # )

    #ans = ans + '\n\nSources: \n' + sources_used
    return ans
```

## 01

Extracting and Wrapping the Answer Text

## 02

Removing the Question Part

## 03

Eliminating Duplicate Sentences

## 04

Returning the Final Answer Text

# Conclusions

1. **Accuracy and Runtime:** 92%. Runtime about 10-15s.

2. **Choice of Dataset:** All the pdf files from EE6405 course slides.
   ( The source pdf can be changed to meet different purposes)

3. **Framework:** We tried wizardlm (87%), llama2-7b-chat (83%), llama2-13b-chat (85%), mistral-7B (92%)

4. **Evaluation Metrics:** Accuracy, Intent Recognition Rate, Task Completion Rate:

# Thank you
# For Listening!

# Reference:


- https://python.langchain.com/docs/get_started/introduction
- https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
- https://www.gradio.app/guides/quickstart
- https://huggingface.co/daryl149/llama-2-13b-chat-hf
- Choi, H., Kim, J., Joe, S., & Gwon, Y. (2021). Evaluation of BERT and ALBERT Sentence Embedding Performance on Downstream NLP Tasks. 2020 25TH INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION (ICPR), 5482–5487. https://doi.org/10.1109/ICPR48806.2021.9412102
- Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,Nils Reimers, Iryna Gurevych,27 Aug 2019