**CHAPTER 2 :**

model.py

*import sys, os*

*import logging*

*import json*

*from llama\_index import SimpleDirectoryReader, VectorStoreIndex, load\_index\_from\_storagefrom llama\_index.storage.storage\_context import StorageContext*

*from llama\_index.indices.service\_context import ServiceContext*

*from llama\_index.llms import OpenAI*

*from llama\_index.node\_parser import SimpleNodeParser*

*from llama\_index.node\_parser.extractors import ( MetadataExtractor, SummaryExtractor, QuestionsAnsweredExtractor, TitleExtractor, KeywordExtractor, )*

*from llama\_index.text\_splitter import TokenTextSplitter  
from dotenv import load\_dotenv*

*import openai*

*import gradio as gr*

*#loads dotenv lib to retrieve API keys from .env file*

*load\_dotenv()*

*openai.api\_key = '' # Instead we can set the Open AI API key in the environment variables which will be used by default*

*# enable INFO level logging*

*logging.basicConfig(stream=sys.stdout, level=logging.INFO)*

*logging.getLogger().addHandler(logging.StreamHandler(stream=sys.stdout))*

*#specify the llm to be used*

*llm = OpenAI(temperature=0.2, model\_name="gpt-3.5-turbo", max\_tokens=512)*

*service\_context = ServiceContext.from\_defaults(llm=llm)*

*#Using the text splitter from llama index to split texts in the documents into chunks*

*text\_splitter = TokenTextSplitter(separator=" ", chunk\_size=1000, chunk\_overlap=200)*

*#set the global service context object and avoid passing service\_context when building the #index*

*from llama\_index import set\_global\_service\_context*

*set\_global\_service\_context(service\_context)*

*#Utilize the metadata extractor class from the llama index for efficient retrieval metadata\_extractor = MetadataExtractor( extractors=[ TitleExtractor(nodes=5, llm=llm), QuestionsAnsweredExtractor(questions=5, llm=llm), SummaryExtractor(summaries=["prev", "self"], llm=llm), KeywordExtractor(keywords=10, llm=llm) ],)*

*#create the node parser to parse nodes from document*

*node\_parser = SimpleNodeParser( text\_splitter=text\_splitter, metadata\_extractor=metadata\_extractor, )*

*#load the pdf paper from source folder*

*document = SimpleDirectoryReader(input\_files=["source/Attention.pdf"], filename\_as\_id=True).load\_data()*

*print(f"loaded document with {len(document)} pages")*

*def load\_index():*

*try:*

*#load storage context for saving the index in the disk*

*storage\_context = StorageContext.from\_defaults(persist\_dir="./storage")*

*#load the index from storage if stored already*

*index = load\_index\_from\_storage(storage\_context)*

*logging.info("Index loaded from storage.")*

*except FileNotFoundError:*

*#create a new index if not found for the first time*

*logging.info("Index not found. Creating a new one...")*

*nodes = node\_parser.get\_nodes\_from\_documents(document)*

*print(f"loaded nodes with {len(nodes)} nodes")*

*#print metadata in json format*

*for node in nodes:*

*# Convert metadata to formatted JSON*

*metadata\_json = json.dumps(node.metadata, indent=5)*

*print(metadata\_json)*

*#create the index based on the nodes and service\_context*

*index = VectorStoreIndex(nodes=nodes, service\_context=service\_context)*

*#save the index to the disk*

*index.storage\_context.persist()*

*logging.info("New index created and saved to storage in disk memory of the host")*

*return index*

*def data\_querying(input\_text):*

*#load the saved index*

*index = load\_index()*

*#to query the index with the input text*

*response = index.as\_query\_engine().query(input\_text)*

*return response.response*

*interface = gr.Interface(fn=data\_querying, inputs=gr.components.Textbox(lines=3, label="Enter the question"),*

*outputs="text", title="PDF Extractor")*

*interface.launch(share=False)*