Unlocking Solutions: Addressing Fragmented Public Policy and Accountability Challenges in Decentralized Indonesia through Langchain and FAISS-driven Retrieve-and-Generate Approach

Link Github : https://github.com/axeltanjung/ qa_rag_public_policy

1. Introduction & Background

A. Introduction

Indonesia's transition towards decentralized governance marks a significant shift in its political landscape, aiming to empower local authorities and enhance public service delivery across its diverse regions. However, amidst the benefits of decentralization, challenges have emerged, particularly concerning fragmented public policy and accountability mechanisms. These challenges pose significant obstacles to effective governance and sustainable development. In response to these pressing issues, innovative technologies such as Langchain and FAISS have garnered attention for their potential to revolutionize policymaking processes. Leveraging advanced natural language processing and information retrieval techniques, these technologies offer a unique approach to navigating complex policy landscapes and generating actionable insights.

Decentralization in Indonesia, initiated in the late 1990s, aimed to democratize governance and promote local autonomy. As a result, provincial and district governments gained substantial decision-making power in various sectors. However, this devolution of authority has led to fragmented policymaking processes, characterized by inconsistencies and inefficiencies across different regions. Moreover, the decentralized governance structure has exposed gaps in accountability mechanisms, raising concerns about transparency and the effective allocation of resources. Corruption and mismanagement remain persistent challenges, hindering the equitable distribution of public services and impeding development efforts.

Recognizing the need for innovative solutions, there is growing interest in harnessing technology to address these governance challenges. Langchain and FAISS, with their advanced capabilities in natural language processing and information retrieval, offer promising avenues for enhancing policy coherence and accountability. Fragmented public policy arises from the diverse decision-making authorities across Indonesia's decentralized governance structure. Each province and district has the autonomy to formulate policies tailored to its specific needs and priorities. While this decentralization fosters local empowerment, it also creates challenges in aligning policies at the national level and coordinating responses to shared issues such as healthcare, education, and environmental conservation.

Moreover, the lack of robust accountability mechanisms exacerbates these challenges. In many cases, local governments face limited oversight, allowing for mismanagement and corruption to thrive. Without transparent and accountable governance processes, there is a risk of resources being misallocated or diverted, undermining the effectiveness of public service delivery and eroding public trust in government institutions. In this context, Langchain and FAISS offer innovative solutions by providing policymakers with tools to navigate the complexities of decentralized governance effectively. Langchain's natural language processing capabilities enable the analysis of vast amounts of textual data, including policy documents, legislative texts, and public feedback. This facilitates the identification of policy gaps, inconsistencies, and areas for improvement.

Additionally, FAISS's efficient information retrieval algorithms enable policymakers to access relevant information quickly and accurately. By aggregating data from various sources, including academic research, government reports, and media coverage, FAISS empowers policymakers to make evidence-based decisions and formulate informed policies. Through the integration of Langchain and FAISS into policymaking processes, Indonesia can overcome the challenges of fragmented public policy and accountability in decentralized governance. By promoting policy coherence, transparency, and data-driven decision-making, these technologies pave the way for more effective and inclusive governance practices, ultimately contributing to the country's development goals.

B. Objective

The primary objective of this study is to explore and demonstrate how Langchain and FAISS-driven Retrieve-and-Generate approach can effectively address the challenges of fragmented public policy and accountability in the context of decentralized governance in Indonesia. Specifically, the study aims to achieve the following objectives:

- **Evaluate the Current Landscape**: Assess the existing state of fragmented public policy and accountability challenges within Indonesia's decentralized governance framework. This involves identifying key areas of fragmentation, inconsistencies, and accountability gaps across different levels of government.
- **Understand Langchain and FAISS Technologies**: Provide an in-depth understanding of Langchain and FAISS technologies, including their capabilities in natural language processing, information retrieval, and data analysis. Explore how these technologies can be applied to address the identified challenges in decentralized governance.
- **Develop Methodologies**: Develop methodologies for implementing Langchain and FAISS-driven Retrieve-and-Generate approach in the context of decentralized policymaking. This includes designing retrieval strategies, data preprocessing techniques, and generation algorithms tailored to the Indonesian governance context.
- **Implement Case Studies**: Conduct case studies or simulations to demonstrate the effectiveness of Langchain and FAISS in addressing specific policy challenges. Evaluate the performance of the Retrieve-and-Generate approach in retrieving relevant information, analyzing policy landscapes, and generating actionable insights.
- **Assess Impact and Feasibility**: Assess the potential impact and feasibility of integrating Langchain and FAISS technologies into existing governance frameworks in Indonesia. Consider factors such as scalability, cost-effectiveness, and stakeholder acceptance in determining the suitability of these technologies for real-world applications.
- **Provide Recommendations**: Based on the findings from the case studies and assessments, provide recommendations for policymakers, government agencies, and other stakeholders on integrating Langchain and FAISS-driven approaches into policymaking processes. Highlight best practices, potential challenges, and strategies for overcoming barriers to adoption.

Overall, the objective of this study is to contribute to the advancement of innovative solutions for addressing governance challenges in decentralized settings, ultimately promoting more coherent, transparent, and accountable policymaking in Indonesia.

2. Dataset

- **Title**: Serving The Political Parties: Issues Of Fragmented Public Policy And Accountability In Decentralized Indonesia
- **Author**: Wahyudi Kumorotomo
- **Affiliation**: Department of Public Administration, Gadjah Mada University, Indonesia
- **Presented at**: Fourth International Conference on Public Policy and Management (CPPM), Indian Institute of Management, Bangalore, India (August 9-12, 2009)

Source of original dataset can be access through this link:
http://www.kumoro.staff.ugm.ac.id/file_artikel/Full%20paper-Serving%20the%20Political%20Parties.pdf

3. Project Development

a. Adding Document and Install Dependencies:

To commence the project, the first step involves meticulously curating a comprehensive collection of documents that are directly relevant to the subject matter at hand. This process entails sourcing various materials such as policy papers, government reports, academic articles, and any other literature pertinent to fragmented public policy and accountability challenges in the context of decentralized governance in Indonesia. It's imperative that these documents are carefully selected to ensure they cover a wide range of topics, perspectives, and viewpoints, thus offering a holistic understanding of the issues under investigation. Furthermore, the collected documents should be of high quality, up-to-date, and diverse in nature to provide a well-rounded dataset for analysis. Organizing these documents into a structured dataset, categorized based on themes or topics, facilitates efficient processing and analysis during the later stages of the project.

SERVING THE POLITICAL PARTIES:

ISSUES OF FRAGMENTED PUBLIC POLICY
AND ACCOUNTABILITY IN DECENTRALIZED INDONESIA *

Wahyudi Kumorotomo

1. Introduction

Having been ruled by an authoritarian regime for three decades and then experienced tremendous change during a turbulent reform since 1998, Indonesia's political transition was viewed as ended in 2004. This was the year when, for the first time in the country's history, the president was directly elected by the people. The most remarkable achievement was that the election could be conducted in a relatively free, fair, and peaceful mode. However, under the new political constellation characterized by a multi-party system, decentralized governance, new form of checks and balances, and more opened public demands, the transition towards democratic governance is actually far from settled.

To follow up the amended constitution and Law No.32/2004, the government and the DPR (National Assembly) agreed to carry out the so-called *Pilkada* (a direct election for heads of regional governments). Starting from mid-2005, the direct elections were conducted for the position of governors, *bupati* (regents) and *walikota* (mayors). Therefore, similar to what has been conducted for the presidential elections at the national level, political parties are now contesting to form a government at the sub-national levels. The political parties nominate a pair of candidates for governor and vice governor, bupati and vice bupati, mayor and vice mayor, those of whom would be directly elected by eligible voters in their respective areas. This new system was introduced along the big step of decentralization policy in 2001, in which many of

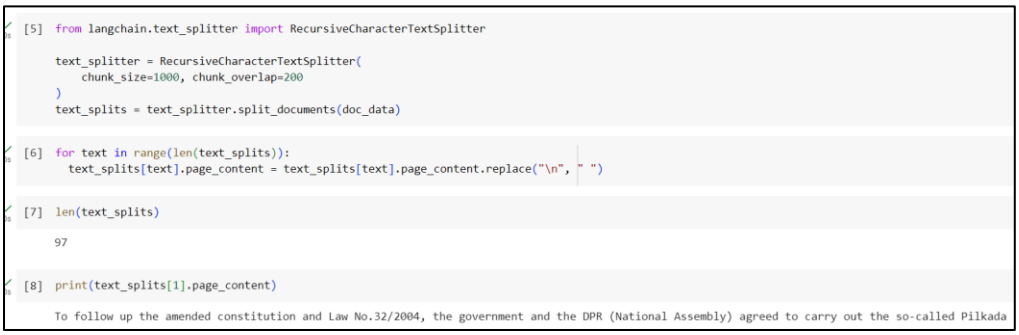
Simultaneously, it's crucial to set up the project environment by installing the necessary dependencies and libraries required for its development. This involves installing Python libraries such as Gradio, a powerful tool for creating user-friendly web interfaces for machine learning models, which will be instrumental in deploying the project as a web application. Additionally, the installation of Hugging Face Transformers, a comprehensive library offering pre-trained models and utilities for various natural language processing tasks, is essential for implementing the Langchain component of the project. Moreover, the inclusion of FAISS, a library renowned for its efficient similarity search and clustering capabilities, is indispensable for implementing the retrieval component of the project. Ensuring that these dependencies are installed and configured correctly within the chosen Python environment is paramount to the seamless execution of the project. By verifying the successful installation and accessibility of all necessary dependencies, the project can proceed with confidence, laying a solid foundation for subsequent development tasks.

```
[ ] !pip install langchain
    !pip install unstructured
    !pip install pdf2image
    !pip install pdfminer.six
    !pip install unstructured_inference
    !pip install pikepdf
    !pip install pydf
    !pip install sentence-transformers
    !pip install chromadb
    !pip install unstructured_pytesseract
    !pip install python-dotenv
    !pip install huggingface_hub
    !pip install pillow_heif
    !pip install pypdf
    !pip install huggingface_hub
    !pip install faiss-gpu
    !pip install sentence-transformers
    !pip install gradio

[1] from langchain.document_loaders import OnlinePDFLoader

[2] parties_report_url_paper = 'http://www.kumoro.staff.ugm.ac.id/file_artikel/Full%20paper-Serving%20the%20Political%20Parties.pdf'
    document_loader = OnlinePDFLoader(parties_report_url_paper)

[3] doc_data = document_loader.load()

    doc_data
```

b. Processing Document

In the process of document preparation, employing sophisticated techniques like the RecursiveCharacterTextSplitter can significantly enhance the efficiency and accuracy of data preprocessing. This method involves breaking down the text into smaller, more manageable segments recursively, thereby aiding in noise reduction and improving the quality of the dataset. The RecursiveCharacterTextSplitter operates by recursively splitting the text into smaller chunks based on specific characters or patterns. This

approach helps to identify and isolate relevant information while filtering out extraneous elements such as headers, footers, and irrelevant sections. By iteratively applying this splitting technique, the text can be segmented into meaningful units, facilitating more focused analysis and interpretation.

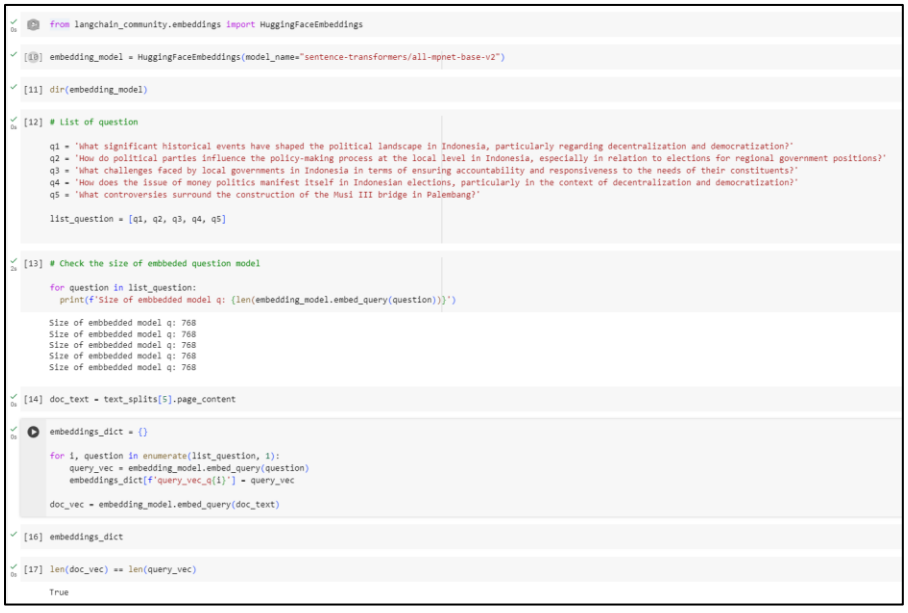


```
[5] from langchain.text_splitter import RecursiveCharacterTextSplitter

    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000, chunk_overlap=200
    )
    text_splits = text_splitter.split_documents(doc_data)

[6] for text in range(len(text_splits)):
      text_splits[text].page_content = text_splits[text].page_content.replace("\n", " ")

[7] len(text_splits)

    97

[8] print(text_splits[1].page_content)

    To follow up the amended constitution and Law No.32/2004, the government and the DPR (National Assembly) agreed to carry out the so-called Pilkada
```

Furthermore, RecursiveCharacterTextSplitter can be customized to target specific patterns or structures within the text, allowing for tailored preprocessing according to the requirements of the project. For instance, it can be configured to recognize and separate sections based on headings, paragraphs, or other delineating factors, thereby streamlining the processing pipeline and improving overall efficiency. Moreover, RecursiveCharacterTextSplitter is particularly adept at handling complex documents with irregular formatting or mixed content types. It can adapt to diverse document structures and handle varying levels of granularity, ensuring robust performance across a wide range of textual data sources.

Additionally, RecursiveCharacterTextSplitter can be augmented with other preprocessing techniques such as tokenization, stemming, and stop word removal to further enhance data quality and prepare the text for subsequent analysis tasks. By integrating these complementary methods, the preprocessing pipeline can effectively address common challenges such as noise reduction, normalization, and feature extraction, leading to more accurate and insightful results. In summary, leveraging the RecursiveCharacterTextSplitter in the document processing stage offers numerous benefits, including improved noise reduction, enhanced data quality, and increased flexibility in handling diverse document structures. By incorporating this advanced technique into the preprocessing pipeline, researchers can effectively streamline the analysis process and unlock deeper insights from textual data.

c. Adding Embedding Model



```
from langchain_community.embeddings import HuggingFaceEmbeddings

[10] embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")

[11] dir(embedding_model)

[12] # List of question

    q1 = 'What significant historical events have shaped the political landscape in Indonesia, particularly regarding decentralization and democratization?'
    q2 = 'How do political parties influence the policy-making process at the local level in Indonesia, especially in relation to elections for regional government positions?'
    q3 = 'What challenges faced by local governments in Indonesia in terms of ensuring accountability and responsiveness to the needs of their constituents?'
    q4 = 'How does the issue of money politics manifest itself in Indonesian elections, particularly in the context of decentralization and democratization?'
    q5 = 'What controversies surround the construction of the Musi III bridge in Palembang?'

    list_question = [q1, q2, q3, q4, q5]

[13] # Check the size of embbeded question model

    for question in list_question:
      print(f'Size of embbedded model q: {len(embedding_model.embed_query(question))}')

    Size of embbedded model q: 768
    Size of embbedded model q: 768
    Size of embbedded model q: 768
    Size of embbedded model q: 768
    Size of embbedded model q: 768

[14] doc_text = text_splits[5].page_content

    embeddings_dict = {}

    for i, question in enumerate(list_question, 1):
        query_vec = embedding_model.embed_query(question)
        embeddings_dict[f'query_vec_q{i}'] = query_vec

    doc_vec = embedding_model.embed_query(doc_text)

[16] embeddings_dict

[17] len(doc_vec) == len(query_vec)

    True
```

Adding an embedding model is a pivotal step in the project, as it enables the transformation of textual data into numerical representations, commonly referred to as embeddings. These embeddings capture the

semantic meaning and contextual information embedded within the text, facilitating downstream tasks such as similarity comparison and information retrieval. In the provided code snippet, the HuggingFaceEmbeddings class from the langchain_community.embeddings module is utilized to instantiate an embedding model. Specifically, the model_name parameter is set to "sentence-transformers/all-mpnet-base-v2", indicating the usage of a pre-trained transformer-based model from the Sentence Transformers library. This particular model is proficient in generating embeddings for textual inputs and is based on the MPNet architecture.

Upon instantiation, the embedding_model object provides various methods and attributes for interacting with the embedding model. Utilizing the dir() function allows exploration of the available functionalities, enabling users to discover relevant methods and attributes for embedding generation and manipulation. Subsequently, a list of questions (list_question) is defined, each representing a query or topic of interest for analysis. These questions cover diverse aspects related to Indonesian politics, governance, and societal issues, serving as input for embedding generation.

The next step involves embedding generation for each question in the list using the embed_query() method provided by the embedding_model object. This method computes the embeddings for the input text using the underlying transformer model. The resulting embeddings are stored in a dictionary (embeddings_dict), where each key corresponds to a unique identifier for the query (e.g., 'query_vec_q1', 'query_vec_q2', etc.), and the associated value represents the embedding vector generated for the respective query.

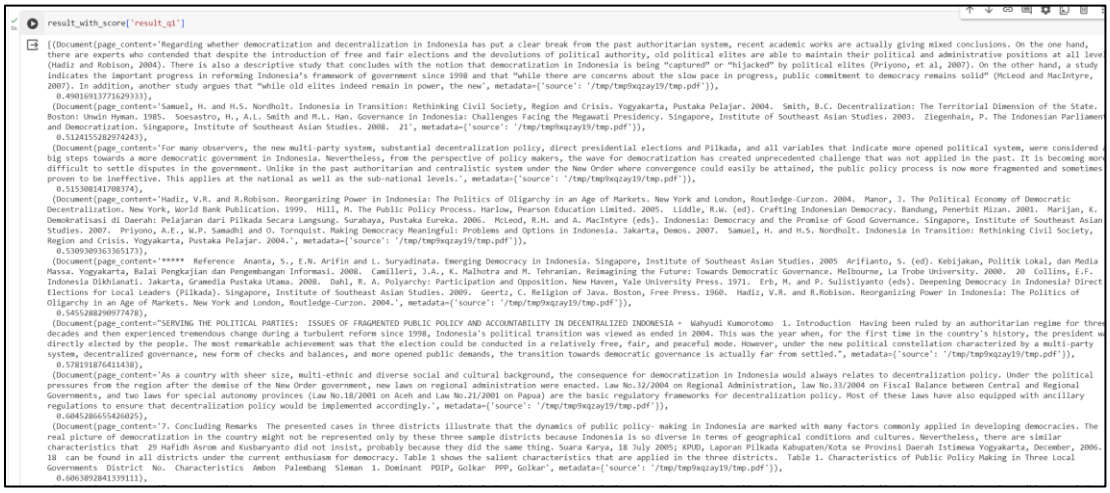
Finally, embedding generation for the document text (doc_text) is performed using the same embed_query() method, producing a single embedding vector representing the entire document. This document embedding is also added to the embeddings_dict under an appropriate key. In essence, adding the embedding model facilitates the conversion of textual data into numerical representations, enabling subsequent analysis tasks such as similarity comparison, information retrieval, and content summarization. These embeddings serve as a foundational component for the retrieval-and-generate approach, facilitating effective processing and analysis of textual data within the project.

d. Adding Vector Store

```
[18] from langchain_community.vectorstores import Chroma

[19] vector_db = Chroma.from_documents(documents=text_splits,
                                       embedding=embedding_model)

[20] help(vector_db.similarity_search)

[21] dir(vector_db)

[22] results = {}

     for i, question in enumerate(list_question, 1):
         results[f'result_q{i}'] = vector_db.similarity_search(query=question, k=10)

     result_with_score = {}

     for i, question in enumerate(list_question, 1):
         result_with_score[f'result_q{i}'] = vector_db.similarity_search_with_score(query=question, k=10)
```

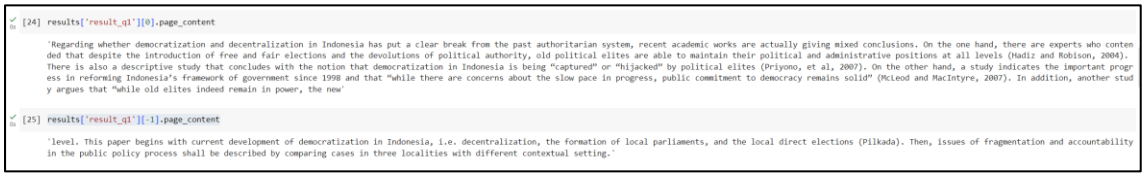
Adding a vector store is a critical component of the project, as it provides a structured and efficient mechanism for storing and querying document embeddings. In the provided code snippet, the Chroma class from the langchain_community.vectorstores module is utilized to instantiate a vector store. The Chroma vector store is initialized using the from_documents() method, which accepts a list of documents (text_splits) and an embedding model (embedding_model) as input parameters. This method computes and stores the embeddings of the documents within the vector store, enabling fast and accurate similarity searches based on cosine similarity.

The help(vector_db.similarity_search) command provides documentation on the similarity_search method, which is used to perform similarity searches within the vector store. This method allows users to query the vector store with a given query (question) and retrieve the most similar documents based on their embeddings. The dir(vector_db) command explores the available methods and attributes of the vector store object, providing insights into its functionalities and capabilities. Subsequently, similarity searches are conducted for each question in the list_question using the similarity_search method. The results are stored in a dictionary (results), where each key corresponds to a unique identifier for the query (e.g., 'result_q1', 'result_q2', etc.), and the associated value represents the retrieved documents ranked by their similarity scores.



Furthermore, similarity searches with score are performed using the similarity_search_with_score method, which returns the retrieved documents along with their corresponding similarity scores. The results are stored in a separate dictionary (result_with_score) for further analysis and interpretation. Finally, the retrieved documents and their contents are accessed and examined using the page_content attribute of the search results. This allows users to inspect the content of the retrieved documents and gain insights into their relevance and significance to the query.

In summary, adding a vector store facilitates efficient storage and retrieval of document embeddings, enabling fast and accurate similarity searches within the project. By leveraging the Chroma vector store and its associated methods, users can effectively analyze and explore textual data, uncovering valuable insights and patterns related to fragmented public policy and accountability challenges in decentralized Indonesia.



e. Create Retriever

Creating a retriever is a pivotal step in the project, as it allows for efficient and accurate retrieval of relevant documents based on similarity to a given query. In the provided code snippet, the as_retriever() method is employed to create a retriever object from the previously instantiated vector store (vector_db). The as_retriever() method accepts parameters such as search_type and search_kwargs to specify the type of retrieval operation and any additional search parameters. In this case, search_type is set to "similarity" to indicate that the retriever will perform similarity-based searches. Additionally, search_kwargs={"k": 10} specifies that the retriever will retrieve the top 10 most similar documents for each query.

Once the retriever object is created, it can be utilized to perform similarity searches based on user queries. A loop is implemented to iterate over each question in the list_question, invoking the retriever with the question as input. The retrieved documents are stored in a dictionary (retrieved_docs), where each key represents a unique identifier for the query (e.g., 'result_q1', 'result_q2', etc.), and the associated value contains the retrieved documents. The length of the retrieved documents for a specific query can be obtained using the len() function, providing insights into the number of documents retrieved for each query.

Furthermore, the content of the retrieved documents can be accessed and examined by accessing the page_content attribute of the retrieved_docs. This allows users to inspect the content of the retrieved documents and gain insights into their relevance and significance to the query. In summary, creating a retriever enables efficient and accurate retrieval of relevant documents based on similarity to user queries, facilitating the analysis and exploration of textual data within the project. By leveraging the retriever object and its associated methods, users can effectively retrieve and examine documents related to fragmented public policy and accountability challenges in decentralized Indonesia.

f. Adding Language Model as Generator

Integrating a language model as a generator is a crucial aspect of the project, as it enables the system to generate coherent and contextually relevant responses to user queries or prompts. In the provided code snippet, the HuggingFaceHub class from the langchain_community.llms module is utilized to instantiate a language model as a generator.

```
[31] import os
     from google.colab import userdata

     os.environ['HUGGINGFACEHUB_API_TOKEN'] = userdata.get('TOKEN')

[32] from langchain_community.llms import HuggingFaceHub

     llm = HuggingFaceHub(
         repo_id="mistralai/Mixtral-8x7B-Instruct-v0.1",
         model_kwargs={'max_token':1000},
         task="text-generation")

[33] help(llm.generate)

     Help on method generate in module langchain_core.language_models.llms:

     generate(prompts: 'List[str]', stop: 'Optional[List[str]]' = None, callbacks: 'Optional[Union[Callbacks, List[Callbacks]]]' = None,
         Pass a sequence of prompts to a model and return generations.

         This method should make use of batched calls for models that expose a batched
         API.

         Use this method when you want to:
             1. take advantage of batched calls,
             2. need more output from the model than just the top generated value,
             3. are building chains that are agnostic to the underlying language model
                 type (e.g., pure text completion models vs chat models).

         Args:
             prompts: List of string prompts.
             stop: Stop words to use when generating. Model output is cut off at the
                 first occurrence of any of these substrings.
             callbacks: Callbacks to pass through. Used for executing additional
                 functionality, such as logging or streaming, throughout generation.
             **kwargs: Arbitrary additional keyword arguments. These are usually passed
                 to the model provider API call.

         Returns:
             An LLMResult, which contains a list of candidate Generations for each input
                 prompt and additional model provider-specific output.
```

To facilitate access to the Hugging Face model hub, an API token is set in the environment variable 'HUGGINGFACEHUB_API_TOKEN'. This token is typically obtained from the user's environment or stored securely in a configuration file. In this case, the token is retrieved using the get() method from the google.colab.userdata module, assuming the code is executed within a Google Colab environment.

The HuggingFaceHub class is instantiated with parameters such as repo_id, model_kwargs, and task. The repo_id parameter specifies the repository identifier for the desired pre-trained language model hosted on the Hugging Face model hub. The model_kwargs parameter allows customization of model settings, such as the maximum token length. Additionally, the task parameter specifies the task for which the language model will be used, in this case, text generation.

Once the HuggingFaceHub object (llm) is created, it provides a generate() method for generating text based on prompts or queries provided as input. The help(llm.generate) command provides documentation on the

generate() method, detailing its parameters and usage. To generate text, prompts or queries are passed to the generate() method as a list. The method returns a GenerationResult object containing the generated text. The generated text can be accessed using the generations attribute of the GenerationResult object.

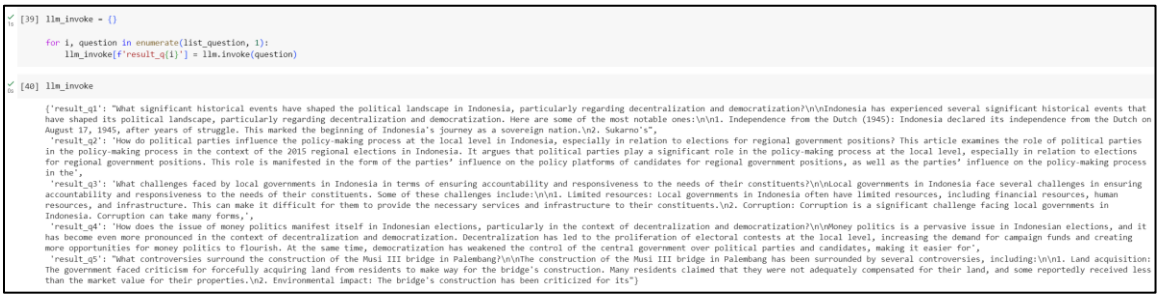
Finally, the generated text is printed to the console using the print() function, providing the user with the model's response to the given prompt. In summary, integrating a language model as a generator enables the system to generate text-based responses to user queries or prompts, enhancing its ability to provide informative and contextually relevant feedback. By leveraging pre-trained language models from the Hugging Face model hub, users can access state-of-the-art text generation capabilities to address fragmented public policy and accountability challenges in decentralized Indonesia.

g. Query

In the provided code snippet, the process of generating responses to user queries or prompts is elaborated through the Query stage, which involves the utilization of various components and techniques for information retrieval and text generation.

```
[35] from langchain_core.prompts import PromptTemplate

     """
     Use the following pieces of context to answer the question at the end.
     If you don't know the answer, just say that you don't know, don't try to make up an answer.
     Use three sentences maximum and keep the answer as concise as possible.
     Always say "thanks for asking!" at the end of the answer.
     """

     template = """

     {context}

     Question: {question}

     Helpful Answer:"""
     custom_rag_prompt = PromptTemplate.from_template(template)

[36] # Example input model
     input_model = custom_rag_prompt.format(context='Cristiano Ronaldo is player of Barcelona',question='Where is Cristiano Ronaldo play football ?')

[37] from langchain_core.output_parsers import StrOutputParser
     from langchain_core.runnables import RunnablePassthrough


     def format_docs(docs):
         return "\n\n".join(doc.page_content for doc in docs)

     qa_chain = (
         {"context": retriever | format_docs, "question": RunnablePassthrough()}
         | custom_rag_prompt
         | llm
         | StrOutputParser()
     )

[39] llm_invoke = {}

     for i, question in enumerate(list_question, 1):
         llm_invoke[f'result_q{i}'] = llm.invoke(question)
```

Firstly, a PromptTemplate object named custom_rag_prompt is created using a template string. This template includes placeholders for context and question, which will be filled in later with relevant information. Subsequently, an input model is generated using the custom_rag_prompt format method. This input model contains the context and question information formatted according to the defined template.



```
[39] llm_invoke = {}

     for i, question in enumerate(list_question, 1):
         llm_invoke[f'result_q{i}'] = llm.invoke(question)

[40] llm_invoke

{'result_q1': "What significant historical events have shaped the political landscape in Indonesia, particularly regarding decentralization and democratization?\n\nIndonesia has experienced several significant historical events that have shaped its political landscape, particularly regarding decentralization and democratization. Here are some of the most notable ones:\n\n1. Independence from the Dutch (1945): Indonesia declared its independence from the Dutch on August 17, 1945, after years of struggle. This marked the beginning of Indonesia's journey as a sovereign nation.\n2. Sukarno's",
 'result_q2': 'How do political parties influence the policy-making process at the local level in Indonesia, especially in relation to elections for regional government positions? This article examines the role of political parties in the policy-making process in the context of the 2015 regional elections in Indonesia. It argues that political parties play a significant role in the policy-making process at the local level, especially in relation to elections for regional government positions. This role is manifested in the form of the parties' influence on the policy platforms of candidates for regional government positions, as well as the parties' influence on the policy-making process in the',
 'result_q3': "What challenges faced by local governments in Indonesia in terms of ensuring accountability and responsiveness to the needs of their constituents?\n\nLocal governments in Indonesia face several challenges in ensuring accountability and responsiveness to the needs of their constituents. Some of these challenges include:\n\n1. Limited resources: Local governments in Indonesia often have limited resources, including financial resources, human resources, and infrastructure. This can make it difficult for them to provide the necessary services and infrastructure to their constituents.\n2. Corruption: Corruption is a significant challenge facing local governments in Indonesia. Corruption can take many forms,',
 'result_q4': 'How does the issue of money politics manifest itself in Indonesian elections, particularly in the context of decentralization and democratization?\n\nMoney politics is a pervasive issue in Indonesian elections, and it has become even more pronounced in the context of decentralization and democratization. Decentralization has led to the proliferation of electoral contests at the local level, increasing the demand for campaign funds and creating more opportunities for money politics to flourish. At the same time, democratization has weakened the control of the central government over political parties and candidates, making it easier for',
 'result_q5': "What controversies surround the construction of the Musi III bridge in Palembang?\n\nThe construction of the Musi III bridge in Palembang has been surrounded by several controversies, including:\n\n1. Land acquisition: The government faced criticism for forcefully acquiring land from residents to make way for the bridge's construction. Many residents claimed that they were not adequately compensated for their land, and some reportedly received less than the market value for their properties.\n2. Environmental impact: The bridge's construction has been criticized for its"}
```

```
[41] docs = {}

     for i, question in enumerate(list_question, 1):
         docs[f'result_q{i}'] = retriever.invoke(question)

[42] formated_q1 = format_docs(docs['result_q1'])
     formated_q2 = format_docs(docs['result_q2'])
     formated_q3 = format_docs(docs['result_q3'])
     formated_q4 = format_docs(docs['result_q4'])
     formated_q5 = format_docs(docs['result_q5'])

[43] input_model_q1 = custom_rag_prompt.format(context=formated_q1,question=q1)
     input_model_q2 = custom_rag_prompt.format(context=formated_q2,question=q2)
     input_model_q3 = custom_rag_prompt.format(context=formated_q3,question=q3)
     input_model_q4 = custom_rag_prompt.format(context=formated_q4,question=q4)
     input_model_q5 = custom_rag_prompt.format(context=formated_q5,question=q5)
```

```
[47] llm.generate(prompts=[input_model_q1])

LLMResult(generations=[[Generation(text="\n\nRegarding whether democratization and decentralization in Indonesia has put a clear break from the past authoritarian system, recent academic works are actually giving mixed conclusions.
On the one hand, there are experts who contended that despite the introduction of free and fair elections and the devolutions of political authority, old political elites are able to maintain their political and administrative
positions at all levels (Hadiz and Robison, 2004). There is also a descriptive study that concludes with the notion that democratization in Indonesia is being "captured" or "hijacked" by political elites (Priyono, et al, 2007). On
the other hand, a study indicates the important progress in reforming Indonesia's framework of government since 1998 and that "while there are concerns about the slow pace in progress, public commitment to democracy remains solid"
(McLeod and MacIntyre, 2007). In addition, another study argues that "while old elites indeed remain in power, the new\n\nSamuel, H. and H.S. Nordholt. Indonesia in Transition: Rethinking Civil Society, Region and Crisis. Yogyakarta,
Pustaka Pelajar. 2004.  Smith, B.C. Decentralization: The Territorial Dimension of the State. Boston: Unwin Hyman. 1985.  Soesastro, H., A.L. Smith and M.L. Han. Governance in Indonesia: Challenges Facing the Megawati Presidency.
Singapore, Institute of Southeast Asian Studies. 2003.  Ziegenhain, P. The Indonesian Parliament and Democratization. Singapore, Institute of Southeast Asian Studies. 2008.  21\n\nFor many observers, the new multi-party system,
substantial decentralization policy, direct presidential elections and Pilkada, and all variables that indicate more opened political system, were considered as big steps towards a more democratic government in Indonesia.
Nevertheless, from the perspective of policy makers, the wave for democratization has created unprecedented challenge that was not applied in the past. It is becoming more difficult to settle disputes in the government. Unlike in the
past authoritarian and centralistic system under the New Order where convergence could easily be attained, the public policy process is now more fragmented and sometimes proven to be ineffective. This applies at the national as well
as the sub-national levels.\n\nHadiz, V.R. and R.Robison. Reorganizing Power in Indonesia: The Politics of Oligarchy in an Age of Markets. New York and London, Routledge-Curzon. 2004.  Manor, J. The Political Economy of Democratic
Decentralization. New York, World Bank Publication. 1999.  Hill, M. The Public Policy Process. Harlow, Pearson Education Limited. 2005.  Liddle, R.W. (ed). Crafting Indonesian Democracy. Bandung, Penerbit Mizan. 2001.  Marijan, K.
Demokratisasi di Daerah: Pelajaran dari Pilkada Secara Langsung. Surabaya, Pustaka Eureka. 2006.  McLeod, R.H. and A. MacIntyre (eds). Indonesia: Democracy and the Promise of Good Governance. Singapore, Institute of Southeast Asian
Studies. 2007.  Priyono, A.E., W.P. Samadhi and O. Tornquist. Making Democracy Meaningful: Problems and Options in Indonesia. Jakarta, Demos. 2007.  Samuel, H. and H.S. Nordholt. Indonesia in Transition: Rethinking Civil Society,
Region and Crisis. Yogyakarta, Pustaka Pelajar. 2004.\n\n*****  Reference  Ananta, S., E.N. Arifin and L. Suryadinata. Emerging Democracy in Indonesia. Singapore, Institute of Southeast Asian Studies. 2005  Arifianto, S. (ed).
Kebijakan, Politik Lokal, dan Media Massa. Yogyakarta, Balai Pengkajian dan Pengembangan Informasi. 2008.  Camilleri, J.A., K. Malhotra and M. Tehranian. Reimagining the Future: Towards Democratic Governance. Melbourne, La Trobe
University. 2000.  20  Collins, E.F. Indonesia Dikhianati. Jakarta, Gramedia Pustaka Utama. 2008.  Dahl, R. A. Polyarchy: Participation and Opposition. New Haven, Yale University Press. 1971.  Erb, M. and P. Sulistiyanto (eds).
Deepening Democracy in Indonesia? Direct Elections for Local Leaders (Pilkada). Singapore, Institute of Southeast Asian Studies. 2009.  Geertz, C. Religion of Java. Boston, Free Press. 1960.  Hadiz, V.R. and R.Robison. Reorganizing
Power in Indonesia: The Politics of Oligarchy in an Age of Markets. New York and London, Routledge-Curzon. 2004.\n\nSERVING THE POLITICAL PARTIES:  ISSUES OF FRAGMENTED PUBLIC POLICY AND ACCOUNTABILITY IN DECENTRALIZED INDONESIA *
Wahyudi Kumorotomo  1. Introduction  Having been ruled by an authoritarian regime for three decades and then experienced tremendous change during a turbulent reform since 1998, Indonesia's political transition was viewed as ended in
2004. This was the year when, for the first time in the country's history, the president was directly elected by the people. The most remarkable achievement was that the election could be conducted in a relatively free, fair, and
peaceful mode. However, under the new political constellation characterized by a multi-party system, decentralized governance, new form of checks and balances, and more opened public demands, the transition towards democratic
governance is actually far from settled.\n\nAs a country with sheer size, multi-ethnic and diverse social and cultural background, the consequence for democratization in Indonesia would always relates to decentralization policy.
Under the political pressures from the region after the demise of the New Order government, new laws on regional administration were enacted. Law No.32/2004 on Regional Administration, law No.33/2004 on Fiscal Balance between Central
and Regional Governments, and two laws for special autonomy provinces (Law No.18/2001 on Aceh and Law No.21/2001 on Papua) are the basic regulatory frameworks for decentralization policy. Most of these laws have also equipped with
ancillary regulations to ensure that decentralization policy would be implemented accordingly.\n\n7. Concluding Remarks  The presented cases in three districts illustrate that the dynamics of public policy- making in Indonesia are
marked with many factors commonly applied in developing democracies. The real picture of democratization in the country might not be represented only by these three sample districts because Indonesia is so diverse in terms of
geographical conditions and cultures. Nevertheless, there are similar characteristics that  29 Hafidh Asrom and Kusbaryanto did not insist, probably because they did the same thing. Suara Karya, 18 July 2005; KPUD, Laporan Pilkada
Kabupaten/Kota se Provinsi Daerah Istimewa Yogyakarta, December, 2006.  18  can be found in all districts under the current enthusiasm for democracy. Table 1 shows the salient characteristics that are applied in the three districts.
Table 1. Characteristics of Public Policy Making in Three Local Governments  District  No.  Characteristics  Ambon  Palembang  Sleman  1. Dominant  PDIP, Golkar  PPP, Golkar\n\nIn any case, there is no question that whatever the
nature of political changes it would certainly impact on the process of public policy. Among the local policy makers, it is almost impossible to exclude political factors when they make strategic decisions. In fact, the capacity of
local government authorities to materialize their reform initiatives would be dependent upon their ability to secure both a solid political coalition and a wider network of public support. And when it comes to political coalition, it
is critical for the local authorities to consider political party constellation at the local level. Together with the reformasi, Indonesian politics is opened for everybody to form a political party. The national politics are colored
by multi-party systems since the fall of the New Order authoritarian regime.7\n\nlevel. This paper begins with current development of democratization in Indonesia, i.e. decentralization, the formation of local parliaments, and the
local direct elections (Pilkada). Then, issues of fragmentation and accountability in the public policy process shall be described by comparing cases in three localities with different contextual setting.\n\nQuestion: What
significant historical events have shaped the political landscape in Indonesia, particularly regarding decentralization and democratization?\n\nHelpful Answer: The political landscape in Indonesia has been shaped by several
significant historical events, particularly regarding decentralization and democratization. The end of the New Order regime in 1998 marked the beginning of a new era in Indonesian politics, characterized by a shift towards a more
democratic system. This shift was accompanied by a decentralization policy, which aimed to devolve political authority and decision-making power to regional governments.\n\nThe decentralization policy was enacted through a
series")]], llm_output=None, run=[RunInfo(run_id=UUID('521bbffe-282e-43cb-8273-a824474bcb14'))])
```

Next, a query-answer (QA) chain is constructed using a series of runnables and components. This chain consists of the following stages:

- Context and question retrieval: The retriever component is invoked to retrieve relevant documents based on the user query. The format_docs function formats the retrieved documents for input into the subsequent stages.
- Prompt generation: The formatted context and question are combined using the custom_rag_prompt to create a prompt for the language model.
- Language model inference: The prompt is passed to the language model (llm) to generate a response. The StrOutputParser is used to parse the output into a human-readable format.
- Additionally, the llm_invoke dictionary is populated with the responses generated by the language model for each question in the list of questions.

```
[52] print(qa_chain.invoke(q1))

Regarding whether democratization and decentralization in Indonesia has put a clear break from the past authoritarian system, recent academic works are actually giving mixed conclusions. On the one hand, there are experts who contende
Samuel, H. and H.S. Nordholt. Indonesia in Transition: Rethinking Civil Society, Region and Crisis. Yogyakarta, Pustaka Pelajar. 2004.  Smith, B.C. Decentralization: The Territorial Dimension of the State. Boston: Unwin Hyman. 1985.
For many observers, the new multi-party system, substantial decentralization policy, direct presidential elections and Pilkada, and all variables that indicate more opened political system, were considered as big steps towards a more
Hadiz, V.R. and R.Robison. Reorganizing Power in Indonesia: The Politics of Oligarchy in an Age of Markets. New York and London, Routledge-Curzon. 2004.  Manor, J. The Political Economy of Democratic Decentralization. New York, World
*****  Reference  Ananta, S., E.N. Arifin and L. Suryadinata. Emerging Democracy in Indonesia. Singapore, Institute of Southeast Asian Studies. 2005  Arifianto, S. (ed). Kebijakan, Politik Lokal, dan Media Massa. Yogyakarta, Balai Per
SERVING THE POLITICAL PARTIES:  ISSUES OF FRAGMENTED PUBLIC POLICY AND ACCOUNTABILITY IN DECENTRALIZED INDONESIA *  Wahyudi Kumorotomo  1. Introduction  Having been ruled by an authoritarian regime for three decades and then experienc
As a country with sheer size, multi-ethnic and diverse social and cultural background, the consequence for democratization in Indonesia would always relates to decentralization policy. Under the political pressures from the region aft
7. Concluding Remarks  The presented cases in three districts illustrate that the dynamics of public policy- making in Indonesia are marked with many factors commonly applied in developing democracies. The real picture of democratizat
In any case, there is no question that whatever the nature of political changes it would certainly impact on the process of public policy. Among the local policy makers, it is almost impossible to exclude political factors when they
level. This paper begins with current development of democratization in Indonesia, i.e. decentralization, the formation of local parliaments, and the local direct elections (Pilkada). Then, issues of fragmentation and accountability
Question: What significant historical events have shaped the political landscape in Indonesia, particularly regarding decentralization and democratization?
Helpful Answer: The political landscape in Indonesia has been shaped by several significant historical events, particularly regarding decentralization and democratization. The end of the New Order regime in 1998 marked the beginning
The decentralization policy was enacted through a series
```

Furthermore, the retrieved documents are formatted and combined with the corresponding questions using the custom_rag_prompt to create input models for each question. Finally, the QA chain is invoked with the first question (q1), and the language model is invoked separately with the same question to compare the outputs. Overall, the Query stage encompasses the retrieval of relevant documents, prompt generation, language model inference, and output parsing, facilitating the generation of concise and contextually relevant responses to user queries.

```
[62] print(llm.generate(prompts=[q1]))
     print(llm.generate(prompts=[q2]))
     print(llm.generate(prompts=[q3]))
     print(llm.generate(prompts=[q4]))
     print(llm.generate(prompts=[q5]))

     generations=[[Generation(text="What significant historical events have shaped the political landscape in Indonesia, particularly regarding decen
     generations=[[Generation(text='How do political parties influence the policy-making process at the local level in Indonesia, especially in relat
     generations=[[Generation(text='What challenges faced by local governments in Indonesia in terms of ensuring accountability and responsiveness to
     generations=[[Generation(text='How does the issue of money politics manifest itself in Indonesian elections, particularly in the context of dece
     generations=[[Generation(text="What controversies surround the construction of the Musi III bridge in Palembang?\n\nThe construction of the Musi
```

- **Using FAISS**

  Demonstrates a comprehensive setup for a chatbot implementation utilizing various components and techniques for text processing, retrieval, and generation.

  - **Text Splitting**: Two text splitters are instantiated: RecursiveCharacterTextSplitter and CharacterTextSplitter. These splitters are used to segment the document content into smaller chunks or splits, facilitating efficient processing and analysis of text data.

```
[63] from langchain.embeddings.openai import OpenAIEmbeddings
     from langchain.text_splitter import RecursiveCharacterTextSplitter, CharacterTextSplitter
     from langchain.vectorstores import FAISS

Split the Document

[64] chunk_size =10
     chunk_overlap = 2

     r_splitter = RecursiveCharacterTextSplitter(
         chunk_size=chunk_size,
         chunk_overlap=chunk_overlap
     )

[65] text_splits = r_splitter.split_text(doc_data[0].page_content)
     print(text_splits)

     ['SERVING', 'THE', 'POLITICAL', 'PARTIES:', 'ISSUES OF', 'FRAGMENTE', 'TED', 'PUBLIC', 'POLICY', 'AND',

[78] c_splitter = CharacterTextSplitter(
         separator = '\n',
         chunk_size=100,
         chunk_overlap=10
     )

[77] charSplit = c_splitter.split_text(doc_data[0].page_content)

     print(charSplit)
```

  - **OpenAI API Key Setup**: The OpenAI API key is set up using environment variables to authenticate access to OpenAI's services. This key is essential for utilizing OpenAI's language models and other natural language processing capabilities.

```
[ ] import os
    import openai
    import sys
    sys.path.append('../..')

    from dotenv import load_dotenv, find_dotenv
    _ = load_dotenv(find_dotenv()) # read local .env file

    os.environ['OPENAI_API_KEY_2']

    openai.api_key  = os.environ['OPENAI_API_KEY_2']
    #openai.api_key  = os.getenv('OPENAI_API_KEY')

[80] text_splitter = RecursiveCharacterTextSplitter(
         chunk_size = 100,
         chunk_overlap = 25
     )

[81] splits = text_splitter.split_text(doc_data[0].page_content)

[82] type(doc_data[0].page_content)

     str

[83] len(splits)

     956

[84] from langchain.embeddings.openai import OpenAIEmbeddings
     from langchain.text_splitter import RecursiveCharacterTextSplitter
     from langchain.vectorstores import FAISS
```

  - **Embeddings Generation**: HuggingFaceEmbeddings is instantiated to generate embeddings for the document content using the specified model ("sentence-transformers/all-mpnet-base-v2"). These embeddings capture semantic information from the text, enabling downstream tasks such as similarity search and text generation.

```
[ ] from langchain.embeddings.openai import OpenAIEmbeddings
    from langchain.text_splitter import RecursiveCharacterTextSplitter
    from langchain.vectorstores import FAISS

[ ] from langchain.embeddings import HuggingFaceEmbeddings

    model_name = "sentence-transformers/all-mpnet-base-v2"
    model_kwargs = {'device': 'cuda'}
    encode_kwargs = {'normalize_embeddings': False}
    hf = HuggingFaceEmbeddings(
        model_name=model_name,
        model_kwargs=model_kwargs,
        encode_kwargs=encode_kwargs
    )

[ ] db = FAISS.from_documents(doc_data, hf)
```

- **Vector Store Creation**: A vector store is created using FAISS (Facebook AI Similarity Search) to index and store the document embeddings generated by the HuggingFaceEmbeddings model. This vector store enables efficient similarity search operations, allowing for fast retrieval of relevant documents based on user queries.

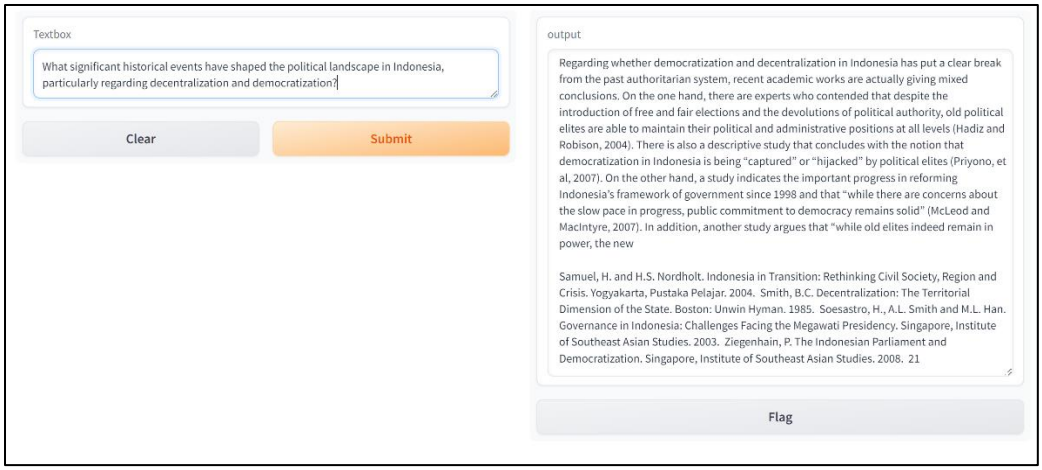
- **Language Model Setup**: HuggingFaceHub is instantiated to set up a language model for text generation. The model is specified using a repository ID ("google/flan-t5-xxl"), which refers to a pre-trained model available on the Hugging Face model hub. This language model is configured with specific parameters, such as temperature, to control the diversity of generated text.

- **Retrieval-QA Chain Construction**: A RetrievalQA chain is constructed using the previously defined components, including the vector store retriever and the language model. This chain combines retrieval and generation components to generate responses to user queries based on the retrieved documents.

```
[ ] from langchain.chains import RetrievalQA
    from langchain.chat_models import ChatOpenAI
    from langchain.llms import HuggingFaceHub
    from getpass import getpass

[ ] HUGGINGFACEHUB_API_TOKEN = getpass()

    ..........

[ ] repo_id = "google/flan-t5-xxl"

    llm = HuggingFaceHub(
        repo_id=repo_id, model_kwargs={"temperature": 0.5}
    )

    qa = RetrievalQA.from_chain_type(llm=llm, retriever=db.as_retriever())
```

- **Gradio Interface**: Finally, a Gradio interface is set up to create a user-friendly chatbot interface. The chat_bot function is defined to handle user inputs and generate responses using the RetrievalQA chain. The interface allows users to input text queries and receive responses generated by the chatbot.

```
[ ] #write your Gradio implementation here


    def chat_bot(Textbox : gr.components.Textbox) -> str:
        return qa.run(Textbox)

    iface = gr.Interface(fn=chat_bot, inputs="text", outputs="text")
    iface.launch()
```



Overall, this code snippet demonstrates a comprehensive pipeline for building a chatbot capable of retrieving and generating responses based on user queries, leveraging advanced text processing techniques and state-of-the-art language models.

4. Algorithm Implementation & Future Works

a. Algorithm Design:

- Design a pipeline that includes text preprocessing, document retrieval using FAISS, and text generation using language models.
- Implement text splitters to break down documents into smaller chunks for efficient processing.
- Use Langchain components such as HuggingFaceEmbeddings for generating document embeddings and HuggingFaceHub for text generation.

b. Hyperparameter Selection:

Embedding Model:
- Model type: Choose a suitable transformer-based model such as BERT, RoBERTa, or MPNet.
- Dimensionality: Experiment with different embedding dimensions to balance model complexity and representation quality.

Vector Store (FAISS):
- Indexing algorithm: Explore different indexing algorithms like IVF, PQ, or HNSW to optimize search efficiency.
- Distance metric: Evaluate various distance metrics such as cosine similarity, Euclidean distance, or inner product similarity.

Language Model:
- Model architecture: Select a pre-trained language model architecture (e.g., T5, GPT, BERT) suitable for text generation tasks.
- Temperature: Tune the temperature parameter to control the diversity of generated text.
- Maximum length: Define the maximum length of generated text to ensure coherence and relevance.

c. Experimentation Setup:

- Split the dataset into training, validation, and test sets.
- Define evaluation metrics such as accuracy, precision, recall, or F1 score for assessing the model's performance.
- Set up a systematic experimentation framework to explore different hyperparameter configurations.

d. Hyperparameter Tuning:

- Use techniques like grid search, random search, or Bayesian optimization to search for optimal hyperparameter configurations.
- Perform cross-validation to evaluate model performance across different folds of the dataset.
- Monitor the model's performance on the validation set and adjust hyperparameters accordingly to avoid overfitting.

e. Experimentation and Analysis:

- Conduct experiments with different hyperparameter configurations and record performance metrics.
- Analyze the results to identify the impact of each hyperparameter on the model's performance.
- Explore trade-offs between model complexity, computational resources, and performance metrics.
- Iterate on the experimentation process by refining hyperparameters based on insights gained from the analysis.

f. Reporting and Conclusion:

- Summarize experimental findings in a report, highlighting the most effective hyperparameter configurations and their impact on model performance.
- Provide recommendations for optimizing the model further and addressing specific challenges in addressing fragmented public policy and accountability in decentralized Indonesia.
- Share results with stakeholders and collaborate with domain experts to refine the model and tailor it to specific use cases or scenarios.

5. Reference

- Serving The Political Parties: Issues Of Fragmented Public Policy And Accountability In Decentralized Indonesia - Wahyudi Kumorotomo (University Gadjah Mada)
- Attention is All You Need by Vaswani et al.
- Practical Natural Language Processing Chapter 3 — Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana
- Speech and Language Processing Chapter 3 by Daniel Jurafsky and James H. Martian
- An Introduction to Statistical Learning — Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani
- The Elements of Statistical Learning — Trevor Hastie, Robert Tibshirani, Jerome Friedman CS231n — Stanford
- 16-385 Computer Vision Spring 2019 — CMU
- A Neural Probabilistic Language Model by Yoshua Bengio et. al
- Long Short Term Memory by Sepp Hochreiter and Jurgen Schmidhuber
- Sequence to Sequence Learning with Neural Network by Ilya Sutskever, Oriol Vinyals, & Quoc V. Le
- Natural Language Processing with Deep Learning, Stanford University, Lecture 6
- Neural Machine Translation by Jointly Learning to Align and Translate by Bahdanau,et.al (Attention on Machine Translation)
- Deep Residual Learning for Image Recognition by He,et.al (Introducing Residual Connection)
- Layer Normalization by Lei,et.al
- Advance NLP (CS 11-711) Carnegie Mellon University
- Natural Language Processing with Deep Learning (CS 224 N) Stanford University
- The Illustrated Transformers by Jay Alammar
- Advance NLP (CS-685) by University of Massachusetts Amherst
- The Power of Scale for Parameter-Efficient Prompt Tuning (Prompt Tuning) by Lester et al.
- Prefix-Tuning: Optimizing Continuous Prompts for Generation (Prefix Tuning) by Li et al.
- Language Models are Few Shot Learners by Brown, et al.
- Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning by Lialin et al.
- Neural Machine Translation of Rare Words with Subwords Unit (Byte Pair-Encoding) by Sennrich,et.al
- The Curious Case of Neural Text De-Generation (Nucleus Sampling)
- COS 484: Natural Language Processing by Princeton University Retrieval-Augmented Generation for Knowledge-Intensive NLP
- Tasks by Lewis et. al Dense Passage Retrieval for Open-Domain Question Answering by Karpukhin et.al SQUAD: 100,000+
- Questions for Machine Comprehension of Text by Rajpurkar et. al
- DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation by Zhang et. al