Task 2 - Optimizing RAG: Innovative Techniques

Technique 1: Dynamic Document Retrieval using Contextual Awareness

Objective

Enhance the precision of document retrieval by incorporating dynamic context awareness, ensuring the retrieved documents are highly relevant to the specific nuances of the user's query.

Implementation

- **Contextual Embedding**: Use advanced contextual embedding techniques such as BERT to capture the semantics of the query.
- **Query Expansion**: Expand the initial query with additional related terms or phrases.
- **Iterative Refinement**: Implement an iterative refinement process where the QA bot re-evaluates and refines the search.

Example Code

python

```
# Example of using BERT for contextual embedding
from transformers import BertTokenizer, BertModel
import torch

# Load pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def get_contextual_embedding(query):
    inputs = tokenizer(query, return_tensors='pt')
    outputs = model(**inputs)
    return outputs.last_hidden_state

query = "What are the company's core values?"
embedding = get_contextual_embedding(query)
print(embedding)
```

Technique 2: Hybrid Response Generation using Ensemble Methods

Objective

Improve the quality and robustness of the generated answers by combining the strengths of multiple models through ensemble methods.

Implementation

- **Model Ensembling**: Combine multiple generative models (e.g., GPT-3, GPT-4) to generate diverse responses.
- **Weighted Averaging**: Use a weighted averaging mechanism to combine the outputs of different models.
- **Response Filtering**: Implement a response filtering system to select the most appropriate answer.

Example Code_Python:

```
# Example of model ensembling
import openai

openai.api_key = "your-openai-api-key"

def generate_response(query):
    response1 = openai.Completion.create(
        model="text-davinci-003",
        prompt=query,
        max_tokens=150
    )
    response2 = openai.Completion.create(
        model="gpt-4",
        prompt=query,
        max_tokens=150
    )
    # Weighted averaging (simple example)
    final_response = (response1.choices[0].text + response2.choices[0].text) / 2
    return final_response.strip()

query = "What are the company's core values?"
response = generate_response(query)
print(response)
```