```
!pip -q install langchain huggingface_hub tiktoken pypdf
```

```
!pip install google-generativeai chromadb
```

Show hidden output

```
!pip install sentence-transformers
```

Show hidden output

```
!pip install langchain_community
```

Show hidden output

```
import os
from google.colab import userdata


GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')
os.environ["GOOGLE_API_KEY"]=GOOGLE_API_KEY
```

```
!pip install langchain_google_genai
```

Show hidden output

```
from langchain_google_genai import ChatGoogleGenerativeAI
llm=ChatGoogleGenerativeAI(model="gemini-1.5-pro")


from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores.chroma import Chroma
import langchain
```

```
from langchain_community.document_loaders import PyPDFLoader


data_path="/content/RAG-FUSION.pdf"


loader=PyPDFLoader(data_path)
docs=loader.load()


docs
```

Show hidden output

```
text_splitter=RecursiveCharacterTextSplitter(chunk_size=500,chunk_overlap=100)


texts = []
for doc in docs:
    texts.extend(text_splitter.split_text(doc.page_content))


texts
```

Show hidden output

```
from langchain.embeddings import HuggingFaceBgeEmbeddings
model_name="BAAI/bge-small-en-v1.5"


encode_kwargs={'normalize_embeddings':True}


embedding_function=HuggingFaceBgeEmbeddings(model_name=model_name,encode_kwargs=encode_kwargs)


db=Chroma.from_texts(texts,embedding_function,persist_directory="./chroma_db")


query="challenges of rag fusion"
```

```
db.similarity_search(query,k=5)
```

```
[Document(metadata={}, page_content='Challenges of RAG-Fusion'),
 Document(metadata={}, page_content='RAG vs RAG-Fusion'),
 Document(metadata={}, page_content='Figure 1: Diagram illustrating the high level process of RAG-Fusion starting with the
original'),
 Document(metadata={}, page_content='that the slowness of RAG-Fusion'),
 Document(metadata={}, page_content='runs shows that RAG-Fusion')]
```

```
retriever=db.as_retriever(k=3)
```

```
retriever.get_relevant_documents(query)
```

```
[Document(metadata={}, page_content='Challenges of RAG-Fusion'),
 Document(metadata={}, page_content='RAG vs RAG-Fusion'),
 Document(metadata={}, page_content='Figure 1: Diagram illustrating the high level process of RAG-Fusion starting with the
original'),
 Document(metadata={}, page_content='that the slowness of RAG-Fusion')]
```

```
from operator import itemgetter
from langchain.prompts import ChatPromptTemplate
from langchain.schema.output_parser import StrOutputParser
from langchain.schema.runnable import RunnablePassthrough,RunnableLambda
```

```
template="""Give detail answer based on the following question
        context:{context}
        question:{question}
        """
```

```
prompt=ChatPromptTemplate.from_template(template)
chain=({"context":retriever,"question":RunnablePassthrough()}
      |prompt
      |RunnableLambda(lambda x: x.messages[0].content)
      |StrOutputParser()
      |llm)
```

```
text_reply=chain.invoke("what is rag vs rag fusion")
print(text_reply)
```

```
eeff91-0' usage_metadata={'input_tokens': 112, 'output_tokens': 102, 'total_tokens': 214, 'input_token_details': {'cache_read': 0}}
```

```
from langchain.prompts import ChatPromptTemplate, SystemMessagePromptTemplate, HumanMessagePromptTemplate, PromptTemplate

prompt = ChatPromptTemplate(
    input_variables=["original_query"],
    messages=[
        SystemMessagePromptTemplate(
            prompt=PromptTemplate(
                input_variables=[],
                template="You are a helpful assistant that generates multiple search queries based on a single input query"
            )
        ),
        HumanMessagePromptTemplate(
            prompt=PromptTemplate(
                input_variables=['original_query'],
                template="Generate multiple search queries related to: {original_query} \n Output('4 queries'):"
            )
        )
    ]
)


original_query="What is MEMS?"
generate_queries=(prompt |llm |StrOutputParser() |(lambda x:x.split("\n")))


from langchain.load import dumps,loads


def reciprocal_rank_fusion(results: list[list], k=60):
  fused_score = {}
  for docs in results:
    for rank, doc in enumerate(docs):
      doc_str = dumps(doc)
```

```
        if doc_str not in fused_score:
          fused_score[doc_str] = 0
        previous_score = fused_score[doc_str]
        fused_score[doc_str] += 1 / (rank + k)
    reranked_result = [(loads(doc), score)
           for doc, score in sorted(fused_score.items(), key=lambda x: x[1], reverse=True)]
    return reranked_result


ragfusion_chain=generate_queries |retriever.map() |reciprocal_rank_fusion
langchain.debug=True


ragfusion_chain.input_schema.schema()
ragfusion_chain.invoke({"original_query":original_query})
```

```
<ipython-input-99-a1fa82eaeb44>:1: PydanticDeprecatedSince20: The `schema` method is deprecated; use `model_json_schema` instead
  ragfusion_chain.input_schema.schema()
[chain/start] [chain:RunnableSequence] Entering Chain run with input:
{
  "original_query": "What is MEMS?"
}
[chain/start] [chain:RunnableSequence > prompt:ChatPromptTemplate] Entering Prompt run with input:
{
  "original_query": "What is MEMS?"
}
[chain/end] [chain:RunnableSequence > prompt:ChatPromptTemplate] [1ms] Exiting Prompt run with output:
[outputs]
[llm/start] [chain:RunnableSequence > llm:ChatGoogleGenerativeAI] Entering LLM run with input:
{
  "prompts": [
    "System: You are a helpful assistant that generates multiple search queries based on a single input query\nHuman: Generate m
  ]
}
[llm/end] [chain:RunnableSequence > llm:ChatGoogleGenerativeAI] [1.18s] Exiting LLM run with output:
{
  "generations": [
    [
      {
        "text": "1. MEMS technology explained\n2. Microelectromechanical systems applications\n3. How MEMS devices work\n4. MEMS
        "generation_info": {
          "finish_reason": "STOP",
          "safety_ratings": []
```

```
      },
      "type": "ChatGeneration",
      "message": {
        "lc": 1,
        "type": "constructor",
        "id": [
          "langchain",
          "schema",
          "messages",
          "AIMessage"
        ],
        "kwargs": {
          "content": "1. MEMS technology explained\n2. Microelectromechanical systems applications\n3. How MEMS devices work\r
          "response_metadata": {
            "prompt_feedback": {
              "block_reason": 0,
              "safety_ratings": []
            },
            "finish_reason": "STOP",
            "safety_ratings": []
          },
          "type": "ai",
          "id": "run-7dfc8661-31c6-4565-b022-30fda7d2447c-0",
          "usage_metadata": {
            "input_tokens": 34,
            "output_tokens": 27,
            "total_tokens": 61,
            "input_token_details": {
              "cache_read": 0
            }
```


```
full_rag_fusion_chain=({"context":ragfusion_chain,"original_query":RunnablePassthrough()}|prompt|llm|StrOutputParser())
```

```
full_rag_fusion_chain.input_schema.schema()
```

```
<ipython-input-106-5ed7e3f3f104>:1: PydanticDeprecatedSince20: The `schema` method is deprecated; use `model_json_schema` instead.
  full_rag_fusion_chain.input_schema.schema()
{'properties': {'original_query': {'title': 'Original Query',
   'type': 'string'},
  'root': {'title': 'Root'}},
 'required': ['original_query', 'root'],
```

```
        'title': 'RunnableParallel<context,original_query>Input',
        'type': 'object'}
```

```
full_rag_fusion_chain.invoke({"original_query":"Give detail MSME"})
```

```
[chain/start] [chain:RunnableSequence] Entering Chain run with input:
{
  "original_query": "Give detail MSME"
}
[chain/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query>] Entering Chain run with input:
{
  "original_query": "Give detail MSME"
}
[chain/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence] Entering Chain
{
  "original_query": "Give detail MSME"
}
[chain/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > prompt:ChatPr
{
  "original_query": "Give detail MSME"
}
[chain/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > prompt:ChatProm
[outputs]
[llm/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > llm:ChatGoogleGe
{
  "prompts": [
    "System: You are a helpful assistant that generates multiple search queries based on a single input query\nHuman: Generate r
  ]
}
[chain/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnablePassthrough] Entering Cha
{
  "original_query": "Give detail MSME"
}
[chain/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnablePassthrough] [0ms] Exiting
{
  "original_query": "Give detail MSME"
}
[llm/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > llm:ChatGoogleGene
{
  "generations": [
    [
      {
        "text": "1. \"MSME definition and explanation\"\n2. \"Detailed explanation of MSME classification (manufacturing, servic
        "generation_info": {
          "finish_reason": "STOP",
          "safety_ratings": []
        },
        "type": "ChatGeneration",
        "message": {
```

```
            "message": {
              "lc": 1,
              "type": "constructor",
              "id": [
                "langchain",
                "schema",
                "messages",
                "AIMessage"
              ],
              "kwargs": {
                "content": "1. \"MSME definition and explanation\"\n2. \"Detailed explanation of MSME classification (manufacturing,
                "response_metadata": {
                  "prompt_feedback": {
                    "block_reason": 0,
                    "safety_ratings": []
                  },
                  "finish_reason": "STOP",
                  "safety_ratings": []
                },
                "type": "ai",
                "id": "run-c02d9637-d10d-497f-9b57-d3bc4c75c170-0",
                "usage_metadata": {
                  "input_tokens": 34,
                  "output_tokens": 49,
                  "total_tokens": 83,
                  "input_token_details": {
                    "cache_read": 0
                  }
                },
                "tool_calls": [],
                "invalid_tool_calls": []
              }
            }
          }
        ]
      ],
      "llm_output": {
        "prompt_feedback": {
          "block_reason": 0,
          "safety_ratings": []
        }
      },
      "run": null,
      "type": "LLMResult"
    }
```

```
}
[chain/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > parser:StrOutp
[inputs]
[chain/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > parser:StrOutput
{
  "output": "1. \"MSME definition and explanation\"\n2. \"Detailed explanation of MSME classification (manufacturing, service)\"
}
[chain/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > chain:Runnable
{
  "input": "1. \"MSME definition and explanation\"\n2. \"Detailed explanation of MSME classification (manufacturing, service)\"
}
[chain/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > chain:RunnableLa
{
  "output": [
    "1. \"MSME definition and explanation\"",
    "2. \"Detailed explanation of MSME classification (manufacturing, service)\"",
    "3. \"MSME registration process and benefits\"",
    "4. \"Government schemes and support for MSMEs\" "
  ]
}
[chain/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > chain:Runnable
{
  "input": [
    "1. \"MSME definition and explanation\"",
    "2. \"Detailed explanation of MSME classification (manufacturing, service)\"",
    "3. \"MSME registration process and benefits\"",
    "4. \"Government schemes and support for MSMEs\" "
  ]
}
[chain/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > chain:RunnableEa
[outputs]
[chain/start] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > chain:reciproc
[inputs]
[chain/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence > chain:reciprocal
[outputs]
[chain/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query> > chain:RunnableSequence] [3.00s] Exiting C
[outputs]
[chain/end] [chain:RunnableSequence > chain:RunnableParallel<context,original_query>] [3.00s] Exiting Chain run with output:
[outputs]
[chain/start] [chain:RunnableSequence > prompt:ChatPromptTemplate] Entering Prompt run with input:
[inputs]
[chain/end] [chain:RunnableSequence > prompt:ChatPromptTemplate] [0ms] Exiting Prompt run with output:
[outputs]
[llm/start] [chain:RunnableSequence > llm:ChatGoogleGenerativeAI] Entering LLM run with input:
```

```
{
  "prompts": [
    "System: You are a helpful assistant that generates multiple search queries based on a single input query\nHuman: Generate r
  ]
}
```


[llm/end] **[chain:RunnableSequence > llm:ChatGoogleGenerativeAI] [1.44s] Exiting LLM run with output:**


```
{
  "generations": [
    [
      {
        "text": "1. \"MSME meaning and definition\"\n2. \"MSME registration process and requirements\"\n3. \"Types of MSMEs and
        "generation_info": {
          "finish_reason": "STOP",
          "safety_ratings": []
        },
        "type": "ChatGeneration",
        "message": {
          "lc": 1,
          "type": "constructor",
          "id": [
            "langchain",
            "schema",
            "messages",
            "AIMessage"
          ],
          "kwargs": {
            "content": "1. \"MSME meaning and definition\"\n2. \"MSME registration process and requirements\"\n3. \"Types of MSM
            "response_metadata": {
              "prompt_feedback": {
                "block_reason": 0,
                "safety_ratings": []
              },
              "finish_reason": "STOP",
              "safety_ratings": []
            },
            "type": "ai",
            "id": "run-febed573-1a68-4bbc-b40d-d330ce5ec5bd-0",
            "usage_metadata": {
              "input_tokens": 41,
              "output_tokens": 44,
              "total_tokens": 85,
              "input_token_details": {
                "cache_read": 0
              }
```

```
                },
                "tool_calls": [],
                "invalid_tool_calls": []
              }
            }
          }
        ]
      ],
      "llm_output": {
        "prompt_feedback": {
          "block_reason": 0,
          "safety_ratings": []
        }
      },
      "run": null,
      "type": "LLMResult"
    }
```


[chain/start] **[chain:RunnableSequence > parser:StrOutputParser] Entering Parser run with input:**