Advanced RAG: Query Rewriting

Presented By: Tahreem Rasul

Overview of Retrieval Augmented Generation (RAG)

What is Retrieval-Augmented Generation (RAG)?

- Combines retrieval of relevant documents with generative models.

Why Use RAG?

- Enhances responses with up-to-date and specific information.
- Mitigates issues like hallucinations in language models.

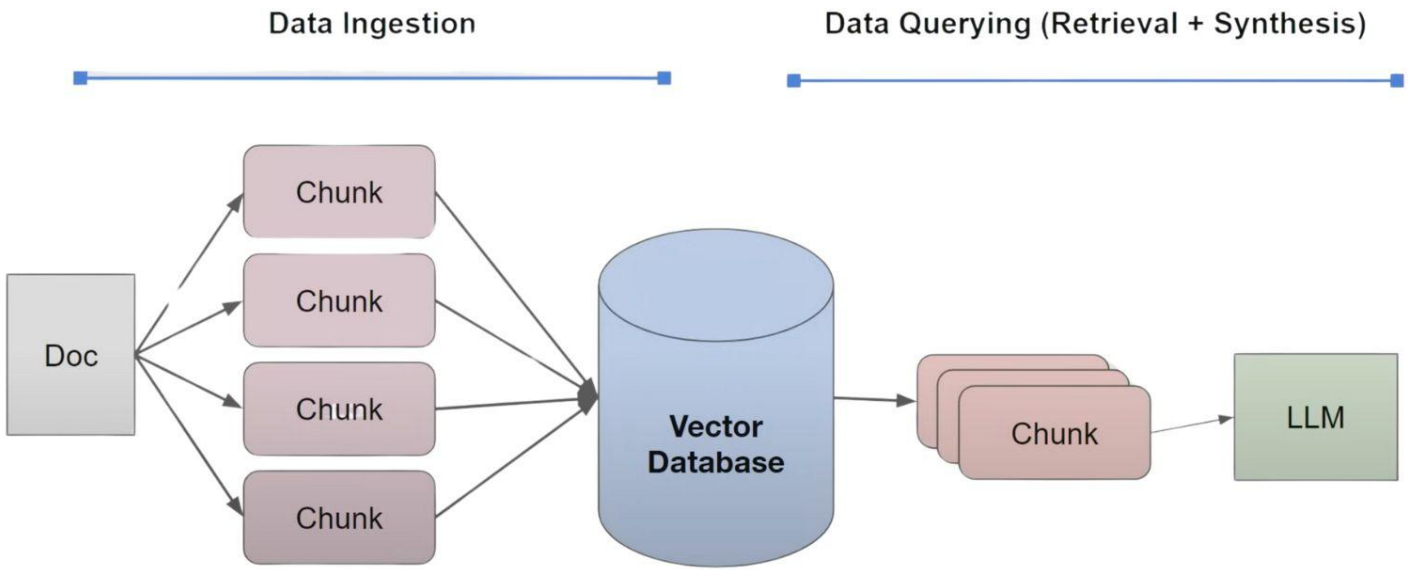
RAG Pipeline

Retrieves relevant documents and loads into working memory/context window

Three Main Components:

1. Indexing/Storage
2. Retrieval
3. Generation

Current RAG Stack for building a Q&A System



Issue with Basic RAG

- RAG retrieves based on query similarity
- Answer may not match query phrasing
- Badly written or differently expressed questions
- Inability to find correct information leads to incorrect answers

Basic Retrieval Is Limited

What is Query Rewriting?

- Rewriting user query for better RAG understanding
- ***Rewrite-retrieve-read*** instead of ***Retrieve-read*** approach
- Uses Generative AI model (large or specially trained)
- Various forms:
    a. Basic
    b. HyDE
    c. Multi-querying
    d. Step-back questions

Why Use Query Rewriting?

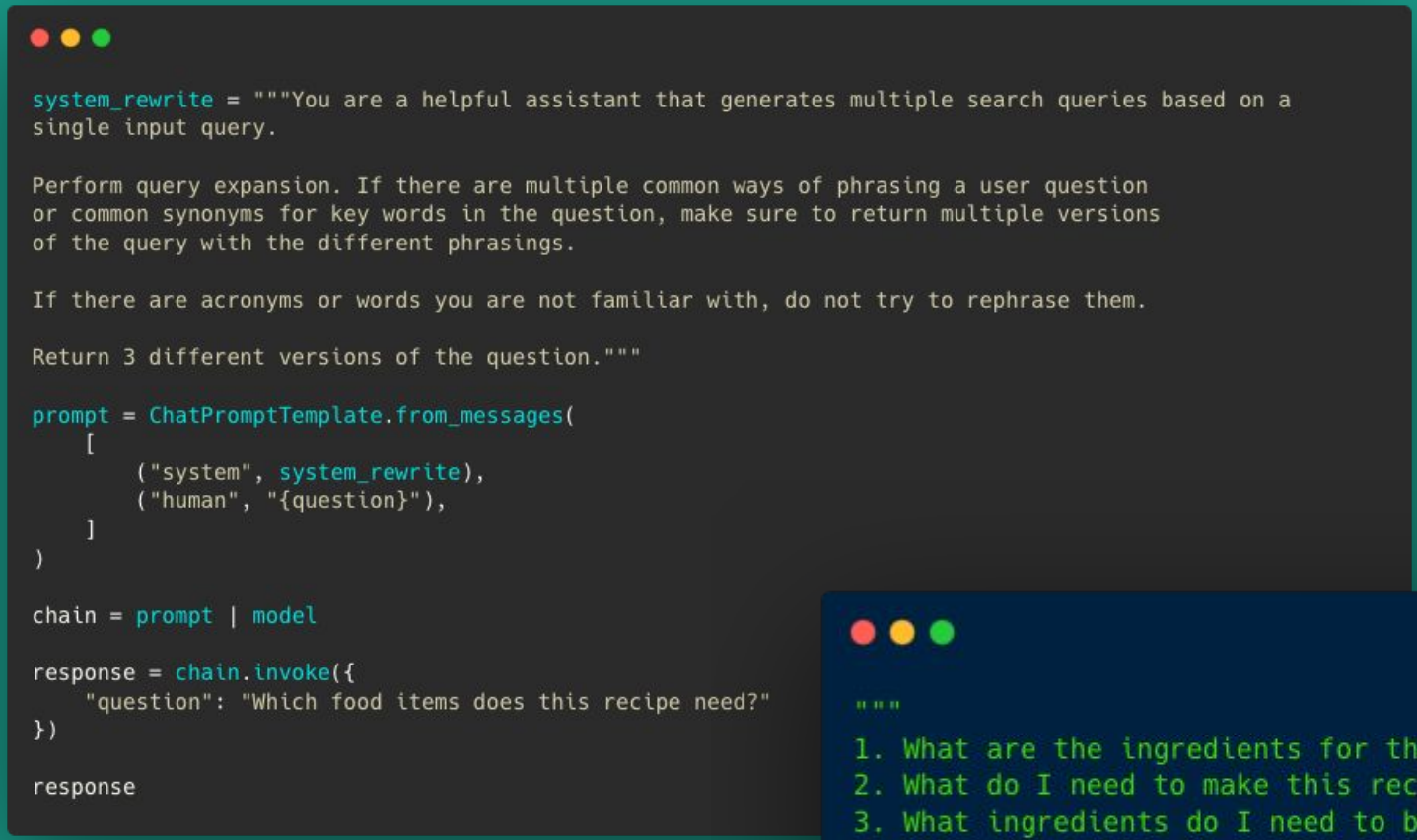
- **R**estructures oddly written questions
- **R**emoves irrelevant context
- **I**ntroduces common keywords
- **S**plits complex questions into sub-questions
- **G**enerates step-back questions for multi-level thinking
- **C**reates hypothetical documents (HyDE) to capture intent

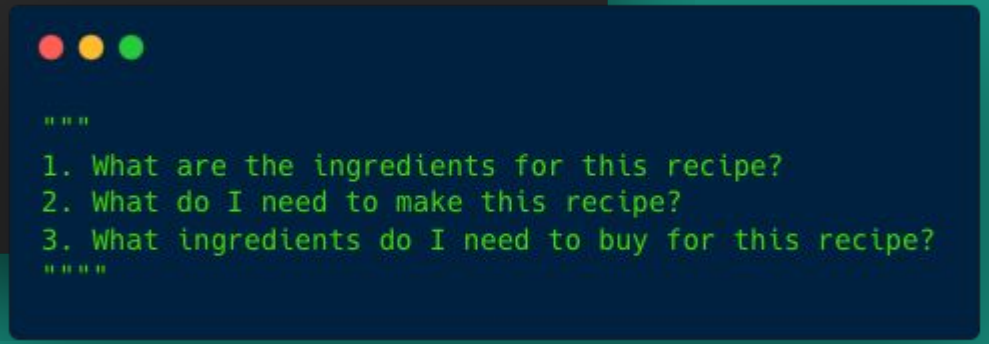
Implementation Strategies

- Zero-shot Query Rewriting
- Few-shot Query Rewriting
- Trainable rewriter
- Sub-queries
- Step-back prompt
- Hypothetical Document Embeddings (HyDE)

Zero Shot Query Rewriting

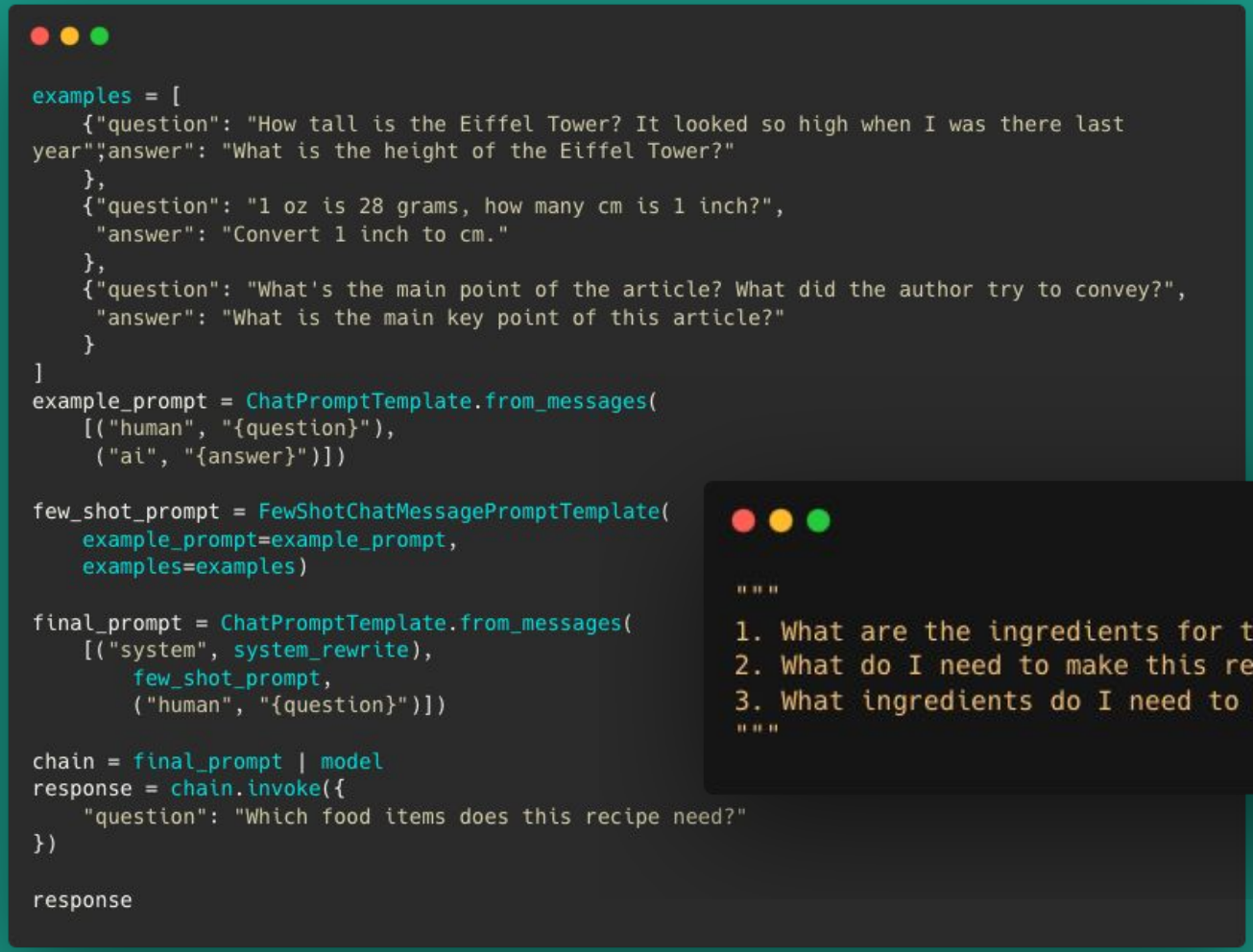
- No examples provided to the LLM
- Simple prompt engineering technique
- LLM rewrites query based on general language understanding
- **Pros:**
  - Quick to implement, requires less prompt engineering
- **Cons:**
  - May be less accurate for complex or domain-specific queries


system_rewrite = """You are a helpful assistant that generates multiple search queries based on a
single input query.
Perform query expansion. If there are multiple common ways of phrasing a user question
or common synonyms for key words in the question, make sure to return multiple versions
of the query with the different phrasings.
If there are acronyms or words you are not familiar with, do not try to rephrase them.
Return 3 different versions of the question."""
prompt = ChatPromptTemplate.from_messages(
[
("system", system_rewrite),
("human", "{question}"),
]
)
chain = prompt | model
response = chain.invoke({
"question": "Which food items does this recipe need?"
})
response


"""
1. What are the ingredients for this recipe?
2. What do I need to make this recipe?
3. What ingredients do I need to buy for this recipe?
""""

Few Shot Query Rewriting

- Provides examples to the LLM of how to rewrite queries
- LLM learns from examples and applies to new queries
- **Pros:**
  - Better performance for specific types of queries
- **Cons:**
  - Requires more tokens per rewrite, careful example curation



```
examples = [
    {"question": "How tall is the Eiffel Tower? It looked so high when I was there last
year","answer": "What is the height of the Eiffel Tower?"
    },
    {"question": "1 oz is 28 grams, how many cm is 1 inch?",
     "answer": "Convert 1 inch to cm."
    },
    {"question": "What's the main point of the article? What did the author try to convey?",
     "answer": "What is the main key point of this article?"
    }
]
example_prompt = ChatPromptTemplate.from_messages(
    [("human", "{question}"),
     ("ai", "{answer}")])

few_shot_prompt = FewShotChatMessagePromptTemplate(
    example_prompt=example_prompt,
    examples=examples)

final_prompt = ChatPromptTemplate.from_messages(
    [("system", system_rewrite),
        few_shot_prompt,
        ("human", "{question}")])

chain = final_prompt | model
response = chain.invoke({
    "question": "Which food items does this recipe need?"
})

response
```

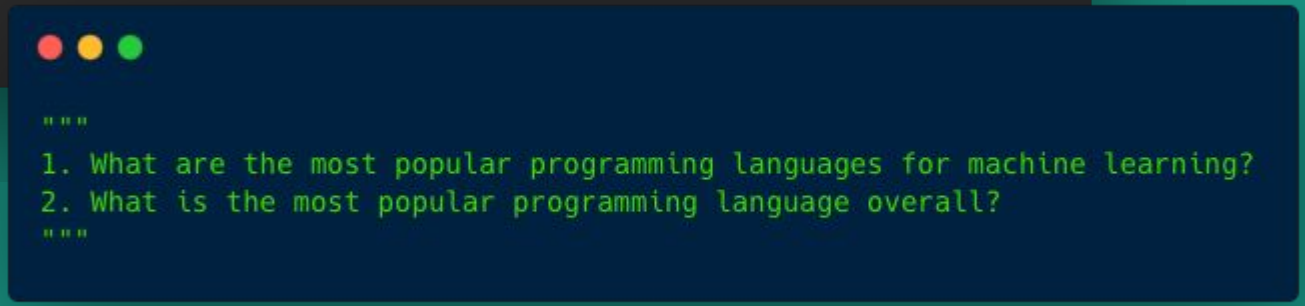


```
"""
1. What are the ingredients for this recipe?
2. What do I need to make this recipe?
3. What ingredients do I need to buy for this recipe?
"""
```

Subqueries

- Decomposes complex queries into multiple simpler sub-queries
- Performs retrieval for each subquery separately
- Useful for multi-part questions or complex information needs
- **Pros:**
  - Improves retrieval for complex queries
- **Cons:**
  - May increase processing time and complexity

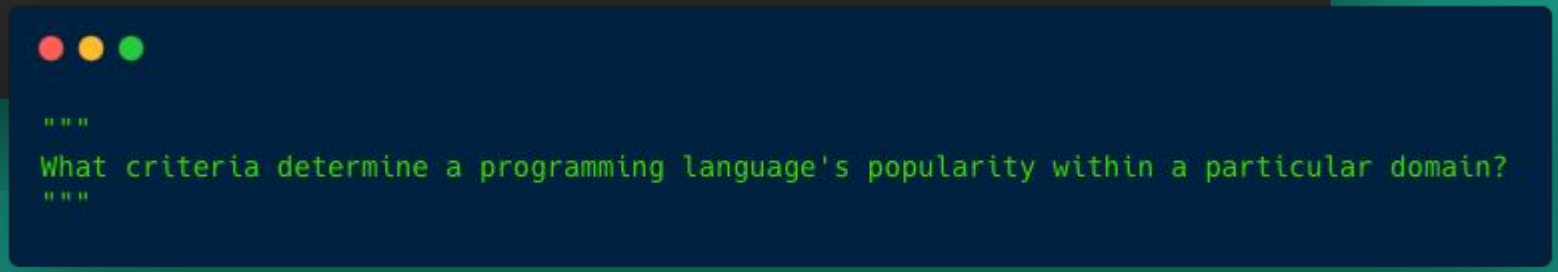
```
system_decompose = """You are a helpful assistant that generates search queries based on a single input
query.

Perform query decomposition. Given a user question, break it down into distinct sub questions that
you need to answer in order to answer the original question.

If there are acronyms or words you are not familiar with, do not try to rephrase them."""

prompt = ChatPromptTemplate.from_messages(
    [("system", system_decompose),
     ("human", "{question}"),
    ])

chain = prompt | model

response = chain.invoke({
    "question": """Which is the most popular programming language for machine learning and
                is it the most popular programming language overall?"""
})

response
```



Step-back Prompt

- Generates more generic, high-level queries
- Helps retrieve relevant information on multiple levels
- Useful for questions requiring multi-level understanding
- **Pros:**
  - Improves context for complex, abstract questions
- **Cons:**
  - May retrieve overly broad information for simple queries

```
system_step_back = """You are an expert at taking a specific question and extracting a more generic
question that gets at
the underlying principles needed to answer the specific question.

Given a specific user question, write a more generic question that needs to be answered in order to
answer the specific question.

If you don't recognize a word or acronym to not try to rewrite it.

Write concise questions."""

prompt = ChatPromptTemplate.from_messages(
    [("system", system_step_back),
      ("human", "{question}"),
    ])

chain = prompt | model

response = chain.invoke({
    "question": """Which is the most popular programming language for machine learning?"""
})

response
```



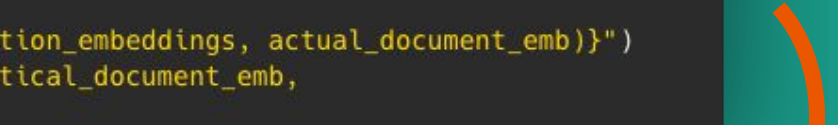
Hypothetical Document Embeddings (HyDE)

- Creates hypothetical context chunks that answer the query
- Uses these to match real context in the database
- Bridges semantic gap between questions and answers
- **Pros:**
  - Improves retrieval when Q&A aren't semantically similar
- **Cons:**
  - Computationally more expensive, may introduce errors

```
actual_document = """
Berkson's paradox, also known as Berkson's bias, collider bias, or Berkson's fallacy, is a result in conditional probability
and statistics which is often found to be counterintuitive, and hence a veridical paradox. It is a complicating factor arising in
statistical tests of proportions. Specifically, it arises when there is an ascertainment bias inherent in a study design. The effect is
related to the explaining away phenomenon in Bayesian networks, and conditioning on a collider in graphical models.

It is often described in the fields of medical statistics or biostatistics, as in the original description of the problem by Joseph Berkson.
"""

system_hyde = """You are an expert at using a question to generate a document useful for answering the question.

Given a question, generate a paragraph of text that answers the question.
"""

prompt = ChatPromptTemplate.from_messages(
    [("system", system_hyde),
     ("human", "{question}"),
    ])

chain = prompt | model

hypothetical_document = chain.invoke({
    "question": """What does Berkson's paradox consist on?"""
})

hypothetical_document
```

```
"""
Berkson's paradox describes a situation where two independent events appear to be negatively correlated due to a biased sampling method. Imagine two desirable traits, like musical talent and good looks, are unrelated in the general population.  However, if you only observe people at a performing arts school, it might seem like the two traits are negatively correlated. This is because individuals with *either* talent or good looks are more likely to be admitted, leading to an overrepresentation of people with one trait but not the other. Essentially, the selection process itself creates a spurious negative correlation by excluding individuals who possess neither or both traits.
"""
```

```python
from sklearn.metrics.pairwise import cosine_similarity

question_embeddings = embeddings_model.embed_documents(["What does Berkson's paradox consist
on?"])
actual_document_emb = embeddings_model.embed_documents([actual_document])
hypothetical_document_emb = embeddings_model.embed_documents([hypothetical_document.content])

print(f"Similarity without HyDE: {cosine_similarity(question_embeddings, actual_document_emb)}")
print(f"Similarity with HyDE: {cosine_similarity(hypothetical_document_emb,
actual_document_emb)}")
```



```
"""
Similarity without HyDE: [[0.86675572]]
Similarity with HyDE: [[0.95113282]]
"""
```