

Special Topics in Data Science

Prompt Engineering: Art and Science for Enhancing QA Interactions  
with Large Language Model

# Final Project Report

Cecilia Nyberg, Yoomin Kim, Zakaria Chahboune  
Group n°12



서울대학교  
SEOUL NATIONAL UNIVERSITY



# Contents

1. Introduction
2. Designing Planning Agent
3. Other Approaches
4. Results and Analysis
5. Conclusion





# Introduction



## Example of structured table

```
Unit: $ in millions
Row: total sales | 2013: $ 44033, 2012: $ 47267, 2011: $ 48047
Row: pharmaceutical | 2013: 37437, 2012: 40601, 2011: 41289
Row: januvia | 2013: 4004, 2012: 4086, 2011: 3324
Row: zetia | 2013: 2658, 2012: 2567, 2011: 2428
Row: remicade | 2013: 2271, 2012: 2076, 2011: 2667
Row: gardasil | 2013: 1831, 2012: 1631, 2011: 1209
Row: janumet | 2013: 1829, 2012: 1659, 2011: 1363
Row: isentress | 2013: 1643, 2012: 1515, 2011: 1359
Row: vytorin | 2013: 1643, 2012: 1747, 2011: 1882
Row: nasonex | 2013: 1335, 2012: 1268, 2011: 1286
Row: proquad/m-m-rii/varivax | 2013: 1306, 2012: 1273, 2011: 1202
Row: singulair | 2013: 1196, 2012: 3853, 2011: 5479
Row: animal health | 2013: 3362, 2012: 3399, 2011: 3253
Row: consumer care | 2013: 1894, 2012: 1952, 2011: 1840
Row: other revenues ( 1 ) | 2013: 1340, 2012: 1315, 2011: 1665
{'column_names': 'column names 2013 2012 2011\n', 'company': 'MRK', 'context_type': 'table', 'fiscal': 2013}
```

# Creating the Database

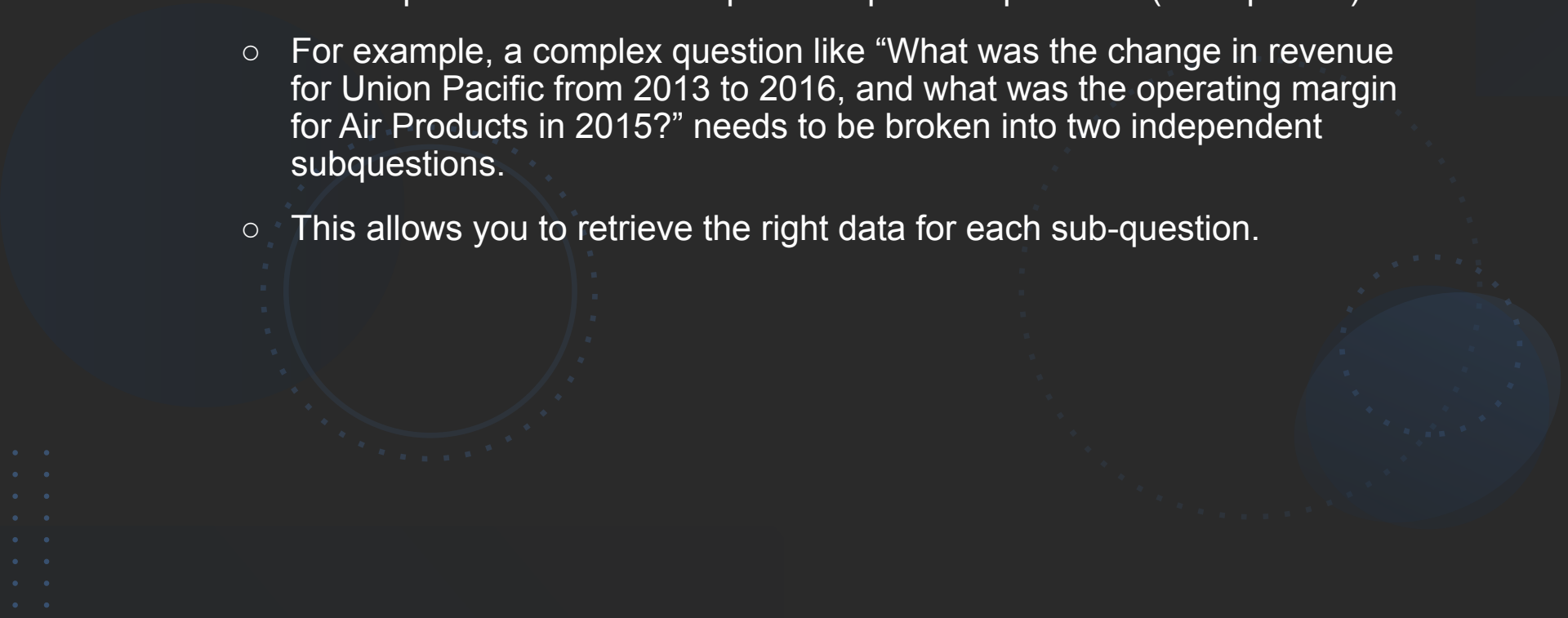
- Structured tables into JSON objects
- Stored pre-text, post-text and structured table in the vectorDB
- Chunk size 1100 overlap 300
- Achieved 25% accuracy with structured approach



# Designing Planning Agent



# Query Decomposition

- Some questions are made up of multiple sub-questions (multiqueries).
  - For example, a complex question like “What was the change in revenue for Union Pacific from 2013 to 2016, and what was the operating margin for Air Products in 2015?” needs to be broken into two independent subquestions.
  - This allows you to retrieve the right data for each sub-question.
- 

# Role of Query Decomposition

- To decompose questions into sub-questions, we utilize a large language model (LLM).
- This LLM is specifically trained to perform the Query Decomposition task and uses a few-shot learning approach to effectively decompose questions by providing concrete examples to the model.
- The decomposed questions are returned in an organized format using the Questions schema (structured output schema).



# Think Tank



- Calculate operating profit margin: Calculate operating profit margin.
- calculate percentage change: Calculate percentage change.
- calculate ratio: Calculate the ratio between two values.
- calculate difference: Calculate the difference between two values.
- rewrite2fiscal: Convert relative time expressions within a question to fiscal years.
- calculate eps: Calculate earnings per share (EPS).
- calculate cashflowfromoperations: Calculate cash flow from operations.



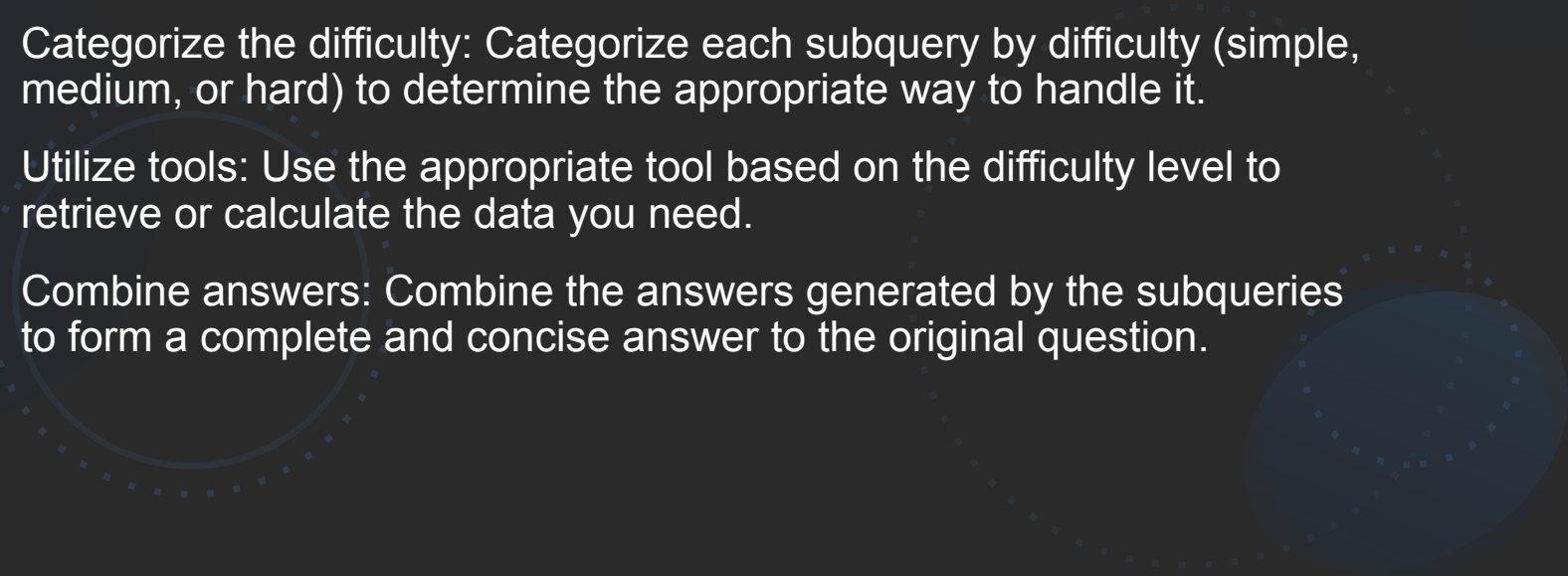
# Role of Think Tank

- Classifying question difficulty:
  - This categorization enables Query Routing, which means you can choose the appropriate tool and search method based on the difficulty level.
- Initial challenges:
  - Initially, we tended to categorize most questions as “high difficulty”.
- Improvements:
  - Added database search capabilities to the Classify Query tool to directly see what data the question requires.



# Chain of Thoughts



- Query decomposition: Break complex questions into smaller subqueries.
  - Categorize the difficulty: Categorize each subquery by difficulty (simple, medium, or hard) to determine the appropriate way to handle it.
  - Utilize tools: Use the appropriate tool based on the difficulty level to retrieve or calculate the data you need.
  - Combine answers: Combine the answers generated by the subqueries to form a complete and concise answer to the original question.
- 

# Retrieval Agent

- Query Expansion
- Reasoning
- CoT

```
system_query = """
Your task is to retrieve important contextual documents to answer the financial query
.
The queries are a bit tricky, so you have to use smart techniques to find the
relevant documents.

Query: {query}.

Use the following strategy:
1. Filter out keywords in the query and use synonyms.
Example:
Query: Air Products current ratio 2024
Keywords:
- current ratio
- liquidity ratio
- cash flow ratio
- working capital ratio
- liquidity
Also, try to keep the query short when retrieving documents.

2. Shorten complex queries.
Example:
Query: What was the Operating Profit Margin of the printing papers business
of International Paper Company in 2005?
Shortened query: Profit printing International Paper Company 2005

3. Conduct the search using the search tools: hybrid_search or retrieve_factual_data.
- If you find a relevant unit in the query or can guess it, search using that.
Example:
Query: What was the Operating Profit Margin of the printing papers business
of International Paper Company in 2005?
Unit query: Paper Company 2005 million dollars $

4. Try different approaches until you retrieve the needed context.
- You can also try using the previous or next year if relevant.

5. Once you find the context, retrieve the relevant documents.

For your help, you can call tools.
Output: {{Relevant context}}
"""
```



# Hybrid Search and Retrieval Improvements

- Hybrid search combining semantic and keyword methods
- Filtered search strategies for improved document retrieval
- Accuracy improved to 54% after adjustments

# Hybrid search

- Uses the BM25Retriever
- Equal weight to both retrievers

```
@tool
def hybrid_search(query: str) -> str:
    """
    Uses hybrid search to retrieve context related documents

    Args:
        query (str): The question to be answered.

    Returns:
        str: The retrieved documents.
    """
    print("using hybrid search")
    print("query", query)
    bm25_retriever = BM25Retriever.from_documents(documents)
    vector_retriever = docsearch.as_retriever()

    retrievers = [bm25_retriever, vector_retriever]
    ensemble_retriever = EnsembleRetriever(retrievers=retrievers, weights=[0.5, 0.5])
    results = ensemble_retriever.get_relevant_documents(query)
    if results:
        # You can combine them however you like. Here, we concatenate them.
        combined_text = ""
        for idx, doc in enumerate(results, start=1):
            combined_text += f"\n[Document {idx}]\n{doc.page_content}\n"
        print("Documents to return: ", combined_text.strip())
        return combined_text.strip()
    else:
        return (
            "No data returned. Try again with a correct ticker/fiscal year, "
            "or adjust your question."
        )
    return result
```

```
user_query Question: Operating Profit Margin of the printing papers business of International Paper Company in 2004
ticker: IP
fy: 2004
question Operating Profit Margin printing papers
ticker IP
fy 2004
question Profit Margin printing papers
ticker IP
fy 2004
question Operating Profit Margin
ticker IP
fy 2004
using hybrid search
query International Paper Company printing papers 2004 profit margin
Documents to return: [Document 1]
printing papers net sales for 2006 decreased 3% ( 3 % ) from both 2005 and 2004 due principally to the sale of the u.s . co

[Document 2]
energy costs ( $ 109 million ) , higher freight costs ( $ 45 million ) and an impairment charge to reduce the carrying value

[Document 3]
. mill operations were favorable compared with 2005 due to current-year improve- ments in machine performance , lower labor
```

# BM25Retriever

- The BM25Retriever uses exact keyword match

$$\text{BM25}(q, d) = \sum_{t \in q} \text{IDF}(t) \cdot \frac{f(t, d) \cdot (k_1 + 1)}{f(t, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)}$$
$$\text{IDF}(t) = \log \left( \frac{N - n(t) + 0.5}{n(t) + 0.5} + 1 \right)$$

# Filtered search

- Search on keyword
- Search on unit
- Filter on tables

```
@tool
def retrieve_factual_data(question: str, ticker: str, fy: int, unit: str = None, keyword: str = None) -> str:
    """
    Search the vector DB for multiple financial reports that match the question, ticker, and fiscal year. Can also search on unit and keyword.

    Args:
        question: The question to be answered
        ticker: The company's ticker symbol for filtering
        fy: The fiscal year for filtering
        unit: The unit for filtering (optional)
        keyword: The keyword for filtering (optional)

    Returns:
        A string containing the combined text of all relevant documents,
        or a message indicating that no data was returned.
    """
    retriever = docsearch.as_retriever(
        search_kwargs={
            'k': 3, # Retrieve more documents for reranking
            'filter': {
                "and": [
                    {"company": {"$eq": ticker}},
                    {"fiscal": {"$eq": fy}}
                ]
            }
        }
    )
    if unit:
        unit_query = query + " " + unit
        unit_table_retriever = docsearch.as_retriever(
            search_kwargs={
                'k': 3, # Retrieve more documents for reranking
                'filter': {
                    "and": [
                        {"context_type": {"$eq": "table"}},
                        {"company": {"$eq": ticker}},
                    ]
                }
            }
        )
        results3 = unit_table_retriever.invoke(unit_query)
    if keyword:
        column_name_table_retriever = docsearch.as_retriever(
            search_kwargs={
                'k': 3, # Retrieve more documents for reranking
                'filter': {
                    "and": [
                        {"context_type": {"$eq": "table"}},
                        {"company": {"$eq": ticker}},
                        {"column_names": {"$eq": keyword}},
                    ]
                }
            }
        )
        results4 = column_name_table_retriever.invoke(question)
```

```
table_retriever = docsearch.as_retriever(
    search_kwargs={
        'k': 2, # Retrieve more documents for reranking
        'filter': {
            "and": [
                {"context_type": {"$eq": "table"}},
                {"company": {"$eq": ticker}},
                {"fiscal": {"$eq": fy}}
            ]
        }
    )
)

results1 = retriever.invoke(question)
results2 = table_retriever.invoke(question)
results = results1 + results2
if unit:
    results = results3 + results
if keyword:
    results = results + results4

if results:
    # You can combine them however you like. Here, we concatenate them.
    combined_text = ""
    for idx, doc in enumerate(results, start=1):
        combined_text += f"\n[Document {idx}]\n{doc.page_content}\n"
    return combined_text.strip()
else:
    return (
        "No data returned. Try again with a correct ticker/fiscal year, "
        "or adjust your question."
    )
```



# Other Approaches



Agent for classifying the queries into levels of complexity.

- More complexity to the model and its implementation
- Less efficient than the Chain-of-thought finally implemented.

```
class CategorizationResponse(BaseModel):
    difficulty_level: int = Field(description="Difficulty level from 1 to 5.")
cat_prompt = PromptTemplate(
    input_variables = ["question"],
    template = """
Categorize the following query into one of the difficulty levels based on its complexity:
- Level 1-2: Simple factual retrieval or straightforward questions.
- Level 3: Questions involving arithmetic operations like "difference" or "average."
- Level 4: Questions requiring comparisons, sums, or intermediate reasoning.
- Level 5: Questions involving complex relationships or multi-step reasoning.

Query: "{question}"

Respond only with the difficulty level number (1, 2, 3, 4, or 5).
"""
)
cat_llm = cat_prompt | llm.with_structured_output(CategorizationResponse)
```

```
def process_query(query: str) -> str:
    difficulty = categorize_question(query)
    if difficulty <= 2:
        return retrieval_chain.invoke(query)
    elif 3 <= difficulty <= 4:
        result = computation_tool(query)
        return f"Computation result: {result}"
    elif difficulty == 5:
        return agent.invoke(query)
```

## Processing time variables

- The first agent: looks for relative time expressions like “last year” and “5 years ago”, and absolute times.
- The second agent: infers the query with the absolute time instead of the relative one.

```
# Step 1: Get today's date
def get_current_year() -> int:
    return datetime.now().year

# Step 2: Define structured response model for temporal alignment
class TemporalAlignmentResponse(BaseModel):
    relative_time: str = Field(description="The relative time, e.g. 'last year' or '6 years ago'. Return 'none' if there is no relative term")
    absolute_year: int = Field(description="The absolute year derived from the query and the relative temporal term if not none.")

temporal_prompt = PromptTemplate(
    input_variables=["query", "current_year"],
    template="""
    Convert relative temporal information in the {query} into absolute years, knowing that the current year is {current_year}.
    - For example, if query contains:
      "5 years ago" : absolute_year = "current_year - 5", relative_time = "in 5 years ago"
      "last year" : absolute_year = "current_year - 1", relative_time = "last year"

    Query: "{query}"

    Respond only with the absolute year.
    """
)

temporal_llm = temporal_prompt | llm.with_structured_output(TemporalAlignmentResponse)
```

```
# Step 4: Temporal alignment function
@tool
def align_temporal_terms(query: str) -> TemporalAlignmentResponse:
    """Aligns temporality for research back in time"""
    current_year = get_current_year()
    response = temporal_llm.invoke({"query": query, "current_year": current_year})
    try:
        return response
    except Exception as e:
        raise ValueError(f"Error parsing response: {response}") from e

# Example usage
example_query = "What is the revenue last year ?"
aligned_temporal = align_temporal_terms(example_query)
print(aligned_temporal.absolute_year)
print(aligned_temporal.relative_time)

example_query = "What is the DISH Network Corporation's current ratio in 13 years ago?"
aligned_temporal = align_temporal_terms(example_query)
print(aligned_temporal.absolute_year)
print(aligned_temporal.relative_time)
```

```
2023
last year
2011
13 years ago
```

# Processing time variables

```
class AbsoluteTimeConverter(BaseModel):
    rewritten_query: str = Field(description="Converts the relative time in the query to an absolute year")

    @tool
    def rewrite_query_with_absolute_time(query: str) -> str:
        """Rewrites the query by converting relative time information to absolute time"""
        try:
            # 1. Extract the relative time component using an LLM
            query_time = align_temporal_terms(query)
            relative_time = query_time.relative_time
            print(relative_time)

            if relative_time.lower() == "none":
                return query # No relative time to convert
```

```
# Example usage:
rewritten_query = rewrite_query_with_absolute_time("What was the revenue of Apple last year?")
print(rewritten_query)

rewritten_query = rewrite_query_with_absolute_time("What was the revenue of Apple in 2022?")
print(rewritten_query)

rewritten_query = rewrite_query_with_absolute_time("What is the DISH Network Corporation's current ratio in 13 years ago?")
print(rewritten_query)
```

```
last year
2023
What was the revenue of Apple last year?
none
What was the revenue of Apple in 2022?
13 years ago
2011
What is the DISH Network Corporation's current ratio in 2011?
```

## # 2. Convert relative time to absolute time using an LLM

```
absolute_year = query_time.absolute_year
print(absolute_year)
```

```
absolute_time_prompt = PromptTemplate(
    input_variables=["query", "relative_time", "absolute_year"],
    template="""
```

You are tasked with converting queries containing relative time expressions into queries with absolute years. Replace the relative time expression '{relative\_time}' in the query '{query}' with the provided absolute year '{absolute\_year}' to ensure clarity and precision. If the query already contains an absolute year, ensure it remains unchanged. Below are examples for guidance:

Examples:

- Query: "What was the revenue of Apple last year?"  
Relative Time: "last year"  
Absolute Year: 2023  
Result: "What was the revenue of Apple in 2023?"



# Results and Analysis

# Results

- 54% of accuracy
- Reason of errors
  - Error propagation: Using multiple LLMs sequentially increases the likelihood of errors spreading within the chain.
  - Unpredictability of LLMs: Due to the nature of LLMs' behavior, it is difficult to always get the expected results.

Table 1: Classification Accuracy

Metric	Value
Accuracy	54%
Correctly Classified Questions	27/50

If you need further assistance or have other questions, please let me know!  
Accuracy: 27/50

# Results

- Correct Answers
  - A total of 26 questions
  - mostly involved simple calculations or simple data retrieval tasks related to a specific year and financial item.
  - For example:
    - “What was American Tower Corp's total lease expense for 2014?” → Correct answer: “\$655.0 million dollars.”
    - “What was BlackRock's long-term component as of December 31, 2011?” → Answer: “\$593.56 million dollars.”

# Results

- Wrong Answers
  - A total of 24 questions
  - For example:
    - Insufficient data: When the database does not have the information you need.
      - “What was the total value of Entergy Corp's long-term securities available for issuance in 2008?”
      - → Unable to answer because the relevant information is not in the database.
    - Complex calculations: Errors occur in complex questions, such as multi-step calculations or ratio calculations.
      - Example: “What was the difference between the operating margins of Air Products in 2015 and International Paper in 2006?”
      - → incorrect results due to inconsistent input data.

# Results

- Wrong Answers
  - A total of 24 questions
  - For example:
    - Question ambiguity: Issues with questions that have a complex structure or unclear keywords.
      - Example: “What is the change in total assets for Goldman Sachs from 2013 to 2017 divided by the sum of the operating margins for Air Products in 2016 and International Paper in 2004?”
      - → confusion in the search and calculation phase.





# Conclusion



# Conclusion



- Utilize pipelines: Process questions and extract relevant information.
- Search the database: Search the database for the data needed to answer the query.
- Reprocess and leverage tools: Reprocess retrieved data based on the nature of the query and leverage agent tools to generate answers.
- Create new pipelines: Develop advanced pipelines and tools for complex cases to continuously improve accuracy.





Thank you