UNIVERSITY OF WEST LONDON
School of Computing and Engineering

Optimized Prompt Engineering Patterns for Credit Score Insight Generation using RAG based Large Language Models

Submitted by:

Kolawole Olanipekun

Student ID: 32147420

Supervisor:
Professor Julie Wall

Date: 29-05-2025

Dedication

This work is first and foremost dedicated to God, whose grace, wisdom, and sustaining strength enabled me to transform ideas into reality.

To my beloved wife (Jane Olanipekun) and son (Asher Olanipekun), thank you for your unwavering support, patience, and the joy you brought me throughout every stage of this journey.

To my dear mother and siblings, your prayers, encouragement, and belief in my potential have been a source of strength.

Finally, this work is also dedicated to all technology practitioners committed to making meaningful contributions and advancing the frontiers of knowledge.

Declaration Page

I, Kolawole Olanipekun, student ID 32147420, hereby declare that this dissertation titled:

"Optimized Prompt Engineering Patterns for Credit Score Insight Generation using RAG-based Large Language Models"

is a result of my original research and work conducted as part of the requirements for the MSc in Software Engineering at the University of West London.

I confirm that all sources consulted and quoted have been appropriately acknowledged in accordance with university guidelines. This dissertation has not been previously submitted in whole or in part for the award of any degree or qualification at this or any other institution.

Signature: ____________________________

Date: 27 May 2025

Acknowledgement

Pursuing this MSc dissertation has been a remarkable journey of intellectual growth and exploration driven by my passion for the application of Artificial Intelligence and a desire to contribute meaningfully to the evolving body of knowledge in this field.

I would like to express my sincere gratitude to Professor Julie Wall, whose guidance, constructive feedback, and critical insights were instrumental in shaping the direction and depth of this research. Her mentorship provided the academic clarity and encouragement needed to see this work through to completion.

I would also like to acknowledge the developers and contributors behind OpenAI's GPT models, whose cutting-edge technology significantly enhanced the research process by enabling rapid access to structured information and reducing the overhead of manual tasks.

To all who played a role directly or indirectly in supporting this academic pursuit, I extend my heartfelt thanks.

Abstract

In the evolving field of financial analytics, credit scoring is fundamental to evaluating borrowers' risk profiles and guiding credit approval decisions. However, traditional credit scoring models often rule-based or built on static machine learning frameworks struggle with adaptability, contextual reasoning, and cross-domain generalization. This dissertation explores a modern approach by leveraging Retrieval-Augmented Generation (RAG) in combination with Large Language Models (LLMs), with a specific focus on optimized prompt engineering patterns for credit score insight generation.

The study begins with a comprehensive Exploratory Data Analysis (EDA) of a credit score dataset sourced from Kaggle, ensuring structural integrity, completeness, and relevance for downstream processing. Two LLMs were evaluated, and GPT-4.0 was selected for its superior contextual reasoning and output accuracy. To enable efficient document retrieval, both FAISS and Weaviate vector databases were evaluated, with Weaviate chosen for its better performance in embedding fidelity and query speed. A modular system architecture was developed to integrate prompt formulation, document retrieval, and LLM response generation through three interconnected pipeline functions. A thorough investigation into prompt engineering was conducted to understand various prompt styles, their respective strengths, and limitations. Based on this analysis, three optimized prompting techniques were adopted: Comparative Prompting with Risk Indicators, Multi-Style Prompting, and Chain-of-Thought Prompting. These techniques were embedded into either of the three distinct prompt formats Custom Prompt Template, Standard Prompt, and Conversation Prompt separately. Initially, ordinary baseline prompts were submitted to the LLM to capture default behaviour. These were subsequently re-engineered using the selected optimization strategies and evaluated using both qualitative metrics (relevance, data accuracy, reasoning quality, clarity, and faithfulness to source) and quantitative analysis using BLEU scores.

The results demonstrate that optimized prompts consistently outperformed baseline prompts in interpretability, factual grounding, and clarity. Notably, the RAG architecture effectively supported LLM response accuracy even with minimal prompt optimization. With optimized prompts achieving an average BLEU score of 0.27 and superior qualitative ratings across all dimensions, this study underscores the transformative potential of combining optimized prompt engineering with RAG-based LLMs for generating transparent, data-driven credit scoring insights.

Table of Contents

Dedication i

Declaration Page ii

Copyright Statement iii

Acknowledgement iv

Abstract v

List of Figures viii

List of Tables ix

Chapter 1 10

1. Introduction 10

1.1 Problem Statement 11

1.2 Research Questions: 11

1.3 Aims and Objectives 12

1.4 Scope of Work 12

Chapter 2 13

2. Literature Review 13

Chapter 3 18

3. Introduction to Prompt Engineering 18

3.2. Key Principles for Crafting Effective Prompts 19

3.3. Advanced Prompt Engineering Techniques 19

3.4. Iterative Prompt Engineering and Refinement 20

3.5. Structured Output Techniques 21

3.6. Prompt Components in RAG Architecture 21

3.7. Evaluation Techniques for RAG-Based Outputs 21

Chapter 4 23

4. Methodology & Approach 23

4.1. Proof of Concept (POC) Development and Workflow 23

4.2. Data Selection 24

4.3. Exploratory Data Analysis 24

4.4. Prompts Questions Generation 25

4.5. Retrieval Augmented Generation 27

4.6.RAG Implementation 28

Document Splitting Strategy 28

4.7. Integrate RAG components with LLM 29

4.7b. Prompt Optimization Strategy 29

4. 8. Embedded LLM Querying without Prompt Optimization 31
4.9. Prompt Test 1 32
4.10. Prompt Test 2 (Not Optimized) 32
4.11. Prompt Test 3 (Not Optimized) 33
4.12. Embedded LLM Querying with Prompt Optimization 33
4.13. First Optimized Prompt 33
4.14. Second Optimized Prompt 35
4.15. Third Optimized Prompt 37
4.16. Evaluation of Output 39
Chapter 5 41
5.Result & Findings 41
5.1. Findings During Exploratory Data Analysis 41
5.2. Prompts Questions Generation 44
5.3. Retrieval Augmented Generation Outputs 44
5.3A. RAG Implementation: Document Loading 44
5.3B. RAG Implementation: Document Splitting Strategy 45
5.3C. RAG Implementation: Document storage using vector databases 46
5.4. Integration of RAG components with LLM 48
5.5. RAG based LLM Querying without prompt Optimization 52
5.6. Prompt Test 1 53
.... 53
5.7. Prompt Test 2 (Not Optimized) 54
5.8. Prompt Test 3 (Not Optimized) 54
5.9. RAG based LLM Querying with prompt Optimization 55
5.10. First Optimized Prompt 55
5.11. Second Optimized Prompt 57
5.12. Third Optimized Prompt 59
5.13. Evaluation of Output 63
Chapter 6 68
6. Conclusion 68
6.1. Ethical Considerations and Risk Management in LLM-Based Financial Systems 68
6.2 Summary of Project Contributions & Critical Analysis 69
6.3. Limitations & Future Work 71
References 72

List of Figures

FIGURE 1: GENERAL CONCEPT BEHIND MESSAGE FLOW IN PROMPT ENGINEERING. SOURCE: (DATACAMP, 2025).......... 18
FIGURE 2: SHOWING THE EFFECT OF HIGH-QUALITY PROMPTS (DATACAMP, 2025).. 19
FIGURE 3: FLOWCHART SHOWING METHODOLOGY WORKFLOW 24
FIGURE 4: FLOWCHART SHOWING WORKFLOW FOR RAG IMPLEMENTATION (DATACAMP, 2025) 28
FIGURE 5A: DATASET LOADED AND PREVIEWED IN NOTEBOOK ENVIRONMENT FOR EDA 41
FIGURE 5B: SNAPSHOT OF THE DATA INSPECTION CARRIED OUT ON CREDIT SCORE DATASET SHOWING NO NULL VALUES . 42
FIGURE 6: STATISTICAL SUMMARY OF CREDIT SCORE DATASET.. 42
FIGURE 7: CHECK FOR UNIQUENESS WITH 259 UNIQUE CREDIT SCORE VALUES 42
FIGURE 8: OBSERVING THE DISTRIBUTION OF THE CREDIT SCORE VARIABLE SHOWING SLIGHT LEFT SKEWNESS............... 43
FIGURE 9: ENCODING OF CATEGORICAL VARIABLES TO CHECK TO CARRY OUT FEATURE IMPORTANCE ANALYSIS 43
FIGURE 10: FEATURE IMPORTANCE ANALYSIS OF PREDICTOR VARIABLES WITH THE DEBT-TO-INCOME RATIO SHOWING HIGHEST IMPORTANCE. 44
FIGURE 11: THE FIRST 5 ROWS LOADED FROM THE CREDIT SCORE DATASET 45
FIGURE 12: THE FIRST 5 ROWS LOADED FROM THE CREDIT SCORE DATASET FOR DOCUMENT LOADING PHASE IN RAG IMPLEMENTATION 45
FIGURE 13: SPLITTING OF CREDIT SCORE DATASET BY SHOWING THE FIRST 100 CHARACTERS OF THE FIRST ROW............ 45
FIGURE 14A: DOCUMENT STORAGE USING FAISS VECTOR DATABASE... 46
FIGURE 14B: DOCUMENT STORAGE USING WEAVIATE VECTOR DATABASE 47
FIGURE 14C: COMPARING OUTPUT OF THE FAISS AND WEAVIATE VECTOR DATABASE WITH WEAVIATE HAVING A BETTER PERFORMANCE. 47
FIGURE 15: FIRST FUNCTION USING GPT 4.0-MINI AND FOUR ARGUMENTS FAISS VECTOR DATABASE 49
FIGURE 16: SECOND FUNCTION USING GPT 4.0 AND FOUR ARGUMENTS AND FAISS VECTOR DATABASE 50
FIGURE 17:THIRD FUNCTION USING GPT 4.0 , ONE ARGUMENT (DOCUMENT) AND WEAVIATE VECTOR DATABASE 51
FIGURE 18: FOURTH FUNCTION USING GPT 4.0 AND THREE ARGUMENTS 52
FIGURE 19: QUALITIVE EVALUATION OF UNOPTIMIZED AND OPTIMIZED PROMPT 1 USING CHATGPT............................ 63
FIGURE 20: MEASUREMENT OF PRECISION FOR RAG BASED LLM RESPONSE PROMPT 1 USING BLEU (OPTIMIZED PROMPTS WITH BLEU SCORE OF 0.3338 & UNOPTIMIZED PROMPTS WITH BLEU SCORE OF 0.2115 64
FIGURE 21: QUALITIVE EVALUATION OF UNOPTIMIZED AND OPTIMIZED PROMPT 2 USING CHATGPT............................ 65
FIGURE 22: MEASUREMENT OF PRECISION FOR RAG BASED LLM RESPONSE PROMPT 2 USING BLEU (OPTIMIZED PROMPTS WITH BLEU SCORE OF 0.3320 & UNOPTIMIZED PROMPTS WITH BLEU SCORE OF 0.1295) 65
FIGURE 23: QUALITIVE EVALUATION OF UNOPTIMIZED AND OPTIMIZED PROMPT 3 USING CHATGPT 66
FIGURE 24: MEASUREMENT OF PRECISION FOR RAG BASED LLM RESPONSE PROMPT 3 USING BLEU (OPTIMIZED PROMPTS WITH BLEU SCORE OF 0.1421 & UNOPTIMIZED PROMPTS WITH BLEU SCORE OF 0.0197) 67
FIGURE 25: BLEU SCORES FOR UNOPTIMIZED AND OPTIMIZED PROMPTS PASS TO THE RAG BASED LLM.................... 71

List of Tables

TABLE 1: CLASSIFICATION OF PROMPTS INTO CATEGORIES AND PATTERNS 15

Chapter 1

1. Introduction

In the finance industry, credit scoring is a key component that helps organizations assess the probability of loan repayment by borrowers and guides decision-making processes related to credit facility approvals. Traditional credit scoring methods are limited by a narrow scope of knowledge and often involve isolated evaluations of credit-related tasks (Feng et al., 2023, p. 1). Existing techniques in financial credit and risk assessment predominantly rely on rule-based or machine learning-driven expert systems, which typically lack adaptability across multiple domains. These approaches are often designed for single-task scenarios and struggle to generalize or integrate insights from unrelated tasks. For instance, a credit scoring model built for a specific objective is often constrained to that use case, limiting its ability to incorporate knowledge from other financial domains.

This fragmentation results in a missed opportunity to leverage insights across related financial activities. For example, anti-fraud techniques developed for the insurance industry could be effectively adapted for use in credit risk assessment. Thus, there is a clear need for a more comprehensive and holistic approach to credit scoring—one that can navigate diverse financial tasks and leverage a broader knowledge base to improve prediction accuracy and model versatility (Feng et al., 2023, p. 2).

According to IBM (2024), Large Language Models (LLMs) are a class of foundational models trained on vast datasets, enabling them to understand and generate human-like language as well as perform various complex tasks. End users interact with LLMs through prompts—carefully crafted textual instructions that guide the model's output. Prompts not only serve as control mechanisms for regulating outputs but also act as a form of soft programming, allowing customization of LLM behavior (White et al., 2023).

The integration of Artificial Intelligence (AI) into financial services has accelerated rapidly, creating transformative opportunities to redefine conventional methodologies. The adoption of LLMs has become increasingly central to enhancing financial decision-making. As financial institutions ingest exponentially growing volumes of data, the need to extract actionable insights from these datasets has become more pronounced.

One of the most significant advancements in this area has been the introduction of Retrieval-Augmented Generation (RAG) architecture. RAG enhances the capabilities of LLMs by enabling them to dynamically retrieve and incorporate external knowledge into their responses. Arslan et al. (2024, p. 1) note that RAG achieves this by integrating document retrieval with natural language generation, thereby improving the accuracy and relevance of generated content. Unlike traditional LLMs that rely solely on pre-trained knowledge, RAG-based systems enrich their outputs by drawing on up-to-date, task-specific external documents, an approach especially valuable in the credit scoring domain, where precision and contextual accuracy are critical.

Prompt engineering has also become an essential skill for maximizing the effectiveness of LLMs. Giray (2023, p. 1) defines prompt engineering as the practice of designing and refining prompts to optimize the performance of LLMs in natural language processing tasks. It is especially relevant in domains like finance, where clarity, accuracy, and bias mitigation are crucial.

This dissertation investigates the impact of optimized prompt engineering patterns for generating credit score insights using RAG-based LLMs. By applying structured prompt templates and contextual document retrieval, this research aims to improve the interpretability and reliability of credit scoring outputs, thereby promoting transparency for financial analysts and end-users alike.

1.1 Problem Statement

Optimized prompt engineering patterns are essential for improving the effectiveness of LLMs in credit scoring applications. Many organizations now utilize LLMs for credit scoring, but without well-structured prompts, outputs can be inaccurate or biased. This undermines the reliability of predictions and could affect financial decision-making. Incorporating optimized prompt patterns improves the performance of LLMs, enhances prediction accuracy, and supports financial stability by providing more dependable credit scores.

1.2 Research Questions:

1. What are the key prompt techniques that formulate effective prompts for credit risk prediction using Retrieval Augmented Generation based Large Language Models
2. How does Retrieval Augmented Generation based Large Language Models respond to optimized prompt requests

1.3 Aims and Objectives

This dissertation aims to optimize prompt engineering patterns to enhance the quality, accuracy, and interpretability of responses generated by Retrieval-Augmented Generation (RAG)-based Large Language Models (LLMs) in the context of credit score insight generation. It critically examines recent advancements in prompt engineering and LLM applications, with a specific focus on refining prompt strategies to improve credit scoring outcomes. To achieve this, recent peer reviewed papers were meticulously picked according to the different approaches used in improving the performance of LLMs with emphasis on either prompt engineering or credit scoring.

This report also aims to address the challenges inherent in credit score insight generation, particularly in the context of leveraging LLMs. By examining existing research and techniques, the aim is to propose optimized prompt engineering patterns that mitigate challenges such as data heterogeneity, interpretability, and response optimization.

1.4 Scope of Work

This chapter presents a high-level introduction to the study, including the problem statement, research questions, objectives, and scope of work. Chapter Two provides a comprehensive literature review of both past and contemporary related studies, with a focus on identifying research gaps. Chapter Three examines various prompt engineering techniques explored in the field. Chapter Four outlines the methodology, system workflow, and architectural design employed in the implementation, with detailed emphasis on Exploratory Data Analysis (EDA), the development of Retrieval-Augmented Generation (RAG), and the pipelines for deploying both standard and optimized prompts using RAG-based Large Language Models (LLMs). Chapter Five highlights results and findings generated through each phase of the workflow. Finally, Chapter Six summarizes the significant project contributions, critical analysis, and recommendations for future research.

Chapter 2

2. Literature Review

Prompting is a brittle process wherein small modifications to the prompt can cause large variations in the model predictions, and therefore significant effort is dedicated towards designing a painstakingly crafted perfect prompt for a task (Arora *et al.,* 2022). Arora *et al* used the methodology of aggregating the predictions of multiple effective, yet imperfect prompts to improve prompting performance over a broad set of models and tasks. The Authors were able to deduce the effectiveness of open-ended questions over restrictive prompts, and the ability to model the varying accuracies and dependencies across a collection of prompts using weak supervision (WS). It is worthy of note to define WS as a powerful framework that learns the accuracy and correlation of multiple noisy sources and aggregates them to produce weak labels for training data (Cachay, Boecking and Dubrawski, 2021). According to Arora et al, one of their key breakthroughs was improving the response of LLM (GPT-J-6B) after getting a new prompt. The major limitation was using a single open source LLM (GPT-J-6B) which questions the extensiveness of this approach when tested with other language models.

Credit and risk assessment is vital in the financial industry, determining the probability of repayment by borrowers, from individuals to nations (Cao, Yang and Yu, 2021). These evaluations, critical for maintaining financial stability, have increasingly moved online. Companies now use online methods for individual assessments like credit scoring and claim analysis to predict default risks (Dastile, Celik and Potsane, 2020). To avoid bias from these online resources, Feng *et al in* 2023 measured the degree of biases in LLMs such as ChatGPT and ChatGPT-4 while considering their effectiveness in credit and risk assessment. Using nine (9) open-source data, these authors deduced that GPT-4 or LLMs fine-tuned with more relevant data have the potential to achieve superior results and potentially replace existing expert systems. However, conducting a bias test, it was discovered that LLMs such as ChatGPT or GPT-4 exhibit some form of bias, affecting individuals' access to online financial services and opportunities. Nevertheless, this research still recorded a significant breakthrough as it was part of the first comprehensive framework for credit scoring using LLMs, encompassing a curated instruction tuning dataset, specialized LLMs, and a set of benchmarks. This framework represents a significant research milestone and a practical toolset applicable within the financial industry to augment the accuracy and depth of credit analysis (Feng *et al.,* 2023). By demonstrating their capacity to comprehend and analyze complex financial data, evidence is provided indicating that LLMs could substantially enhance the accuracy and efficiency of credit scoring practices in the financial industry (Feng *et al.,* 2023).

Attention is also drawn to the ethical considerations inherent in the deployment of LLMs, particularly in sensitive applications such as credit scoring, which carry significant societal ramifications. To foster ethical use and continual innovation, all resources have been open-sourced, facilitating scrutiny, adaptation, and progression of research within both the academic and industrial spheres (Feng *et al.,* 2023). A major downside to the Feng et al project is that these LLMs still experience some form of bias and the prompt patterns used were limited to just two prompt types thus not giving us a full picture of how to measure the effect of prompts patterns on LLMs.

Moving forward, White *et al. In 2023,* highlights a framework for documenting patterns for structuring prompts to solve a range of problems so that they can be adapted to different domains. They presented a catalog of patterns that have been applied successfully to improve the outputs of LLM conversations. Using open-source data and agile methodology, they tested how prompts can be built from multiple patterns and illustrated the benefits of combining prompt patterns. White *et al.* confirmed the influence of a prompt extends to subsequent interactions and output generated from an LLM, as it furnishes specific rules and guidelines governing the conversation. By establishing a predetermined set of initial rules, a prompt delineates the context of the conversation, indicating to the LLM the significance of information and dictating the desired form and content of the output (White *et al.,* 2023).Key research progress includes codifying the knowledge of prompt patterns to enhance reuse and transferability to other context and domains where users face similar but not identical problems, proposition of concepts for fundamental contextual statements, which are written descriptions of the important ideas to communicate in a prompt to an LLM and Classification of Prompts into categories and patterns . Although the prompts are generalizable but most of the demonstrations were construed to the software development domain, giving some doubts of its performance around credit scoring use cases.

Pattern Category	**Prompt Pattern**
Input Semantics	Meta Language Creation

Output Customization	Output Automater Persona Visualization Generator Recipe Template
Error Identification	Fact Check List Reflection
Prompt Improvement	Question Refinement Alternative Approaches Cognitive Verifier Refusal Breaker
Interaction	Flipped Interaction Game Play Infinite Generation
Context Control	Context Manager

Table 1: Classification of prompts into categories and patterns (White *et al.,* 2023)

It is not far-fetched to consider other concepts of determining accurate credit score predictions using other supervised learning methods other than LLMs. Very recently, Babaei and Giudici, 2024 made attempts to improve the performance of large language models to a level like classical **logistic regression** (LR) models while subjecting them to credit scoring applications. The authors concluded that GPT-4 or LLMs fine-tuned with more relevant data have the potential to achieve superior results and potentially replace existing expert systems. However, conducting a bias test, it was discovered that LLMs such as ChatGPT or GPT-4 exhibit biases, affecting individuals' access to online financial services and opportunities (Babaei and Giudici, 2024). LLMs outputs can be improved by informing them about train data but not the whole train data, only using a small sample of it (Babaei and Giudici, 2024). The findings of these authors showed that while logistic regression is simple and most used for binary classification tasks, GPT model which is an LLM application can perform similarly to the classic classifier in a credit decision-making problem. They showed that using only a small sample of train data in the GPT model can produce almost similar results to the regression model. Therefore, users without data science knowledge who intend to use few train observations can handle a credit decision problem and find the creditworthiness of the loan applicants (Babaei and Giudici, 2024). Mentioning limitations, Babaei and Giudic did not lay emphasis on optimized prompt engineering procedures that could boost the model performance (accuracy).

Another recent piece of research though outside the credit scoring bucket, was the prompting of large language models for recommendation systems. **Xu *et al*** *early this year explored public datasets in their research and* divided the general framework of prompt engineering specialized for recommender systems into **four** key elements which are task description (the condition for constructing prompts), user interest modeling (mining and utilization of user interest gives characteristic features to the recommender task, candidate items construction(these affects how the recommender systems are able to provide appropriate items or selection matching for users) and prompting strategies(this is concerned with the reasoning abilities of LLM). This research pointed out that candidate items construction for LLMs is crucial to the final recommendation results (Xu *et al.*, 2024). Also, retrieving candidate items by traditional recommendation models first, and then re-ranking items by LLMs can further improve the results, but the performance varies on specific methods and datasets (Xu *et al.,* 2024). The future direction of this project is to look for efficiency. The key limitation of leveraging LLMs in industrial recommender systems is efficiency (Fan, W. *et al.*, 2023) including considerations of both time and space. On the one hand, the fine-tuning and inference efficiency of LLMs cannot compare to traditional recommendation models while techniques such as parameter-efficient fine-tuning can aid in keeping LLMs updated in a computationally efficient manner, recommender systems need to iterate continuously over time, i.e., incremental learning (Xu *et al.*, 2024).

The term Retrieval-Augmented Generation was first introduced by (Lewis *et al.,* 2020). Large Language Models (LLMs) possess impressive capabilities but still encounter limitations in real-world use, including generating inaccurate information (hallucinations), delayed updates to knowledge, and a lack of transparency in their responses. Retrieval-Augmented Generation (RAG) addresses these issues by retrieving pertinent information from external knowledge sources prior to generating responses with an LLM. This approach has been shown to significantly improve the accuracy of answers and reduce hallucinations, especially in tasks requiring extensive domain knowledge. Moreover, by referencing the sources of information, RAG enables users to validate responses, thereby enhancing trust in the model's outputs(Arslan *et al.,* 2024).

Large Language Models (LLMs) have revolutionized the field of Natural Language Processing (NLP), highlighting a level of linguistic proficiency and knowledge comprehension unprecedented in earlier models(Gao *et al.,* 2023). Advanced systems such as the GPT series, LLaMA, and Gemini models have demonstrated the ability to exceed human performance benchmarks across various evaluation metrics (Gao et al., 2023). Despite these achievements, LLMs still face significant limitations. They are known to generate inaccurate or fabricated information, particularly when responding to domain-specific or highly technical queries that extend beyond their training data or require up-to-date knowledge (Gao et al., 2023). These challenges underscore the risks of using LLMs as black-box systems in production environments where reliability and factual accuracy are critical.

Traditionally, adapting LLMs to new or proprietary domains has involved fine-tuning their parameters to incorporate additional knowledge. While this approach can be effective, it is resource-intensive, costly, and technically complex, making it less practical for continuously evolving information needs (Gao et al., 2023). Within this context, a distinction emerges between parametric and non-parametric knowledge. Parametric knowledge is embedded in the model's internal weights during training and reflects the model's generalization of learned patterns. In contrast, non-parametric knowledge is maintained in external sources—such as vector databases—and can be updated independently of the model. This external knowledge allows LLMs to enhance the relevance and accuracy of their outputs by dynamically incorporating current or specialized information (Gao et al., 2023). However, relying solely on parametric memory presents issues: important but infrequent information may be forgotten, the model's internal knowledge becomes outdated, and scaling the model to capture more knowledge increases training and inference costs (Gao et al., 2023).

Large pre-trained language models are known to encode factual knowledge within their parameters and often achieve state-of-the-art performance when fine-tuned for specific NLP tasks. However, they still face challenges in accurately accessing and manipulating this stored knowledge. As a result, they often underperform compared to specialized architectures on knowledge-intensive tasks. Furthermore, offering traceable sources for their outputs and efficiently updating their internal knowledge base remain significant areas of ongoing research(Lewis *et al.*, 2020)

Chapter 3

3. Introduction to Prompt Engineering

Prompt engineering is the process of crafting effective input prompts to guide large language models (LLMs) like ChatGPT to generate desired outputs. This technique is analogous to preparing a recipe, where the quality and composition of the ingredients (i.e., the prompt) determine the quality of the final dish (i.e., the response). Clear, structured, and high-quality prompts tend to produce more accurate and relevant model outputs, while vague or ambiguous prompts may lead to suboptimal results (Datacamp, 2025).

The OpenAI API provides developers with tools to experiment with different prompt types and control parameters such as temperature (which governs randomness) and max_tokens (which limits output length). To simplify repeated API interactions, a reusable function like get_response(prompt) can be defined, allowing users to submit a prompt and receive a model-generated response (DataCamp, 2025).

Below is a simple framework behind message flow in prompt engineering, followed by another diagram from Datacamp learning platform that recognizes the effect of high-quality prompt.

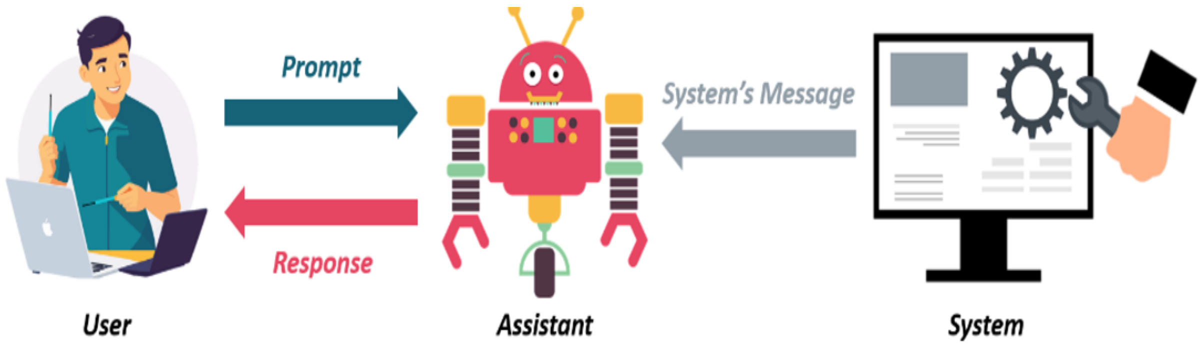

Figure 1: General concept behind message flow in prompt engineering. Source: (DataCamp, 2025).

Figure 2: Showing the effect of High-Quality prompts (DataCamp, 2025).

3.2. Key Principles for Crafting Effective Prompts

Effective prompt engineering relies on several best practices:

I. Use of Action Verbs: Action-oriented verbs like "write," "explain," or "describe" guide the model clearly. Avoid ambiguous terms such as "understand" or "feel".

II. Specific Instructions: Detailed instructions about context, format, tone, length, and target audience yield more tailored responses.

III. Delimited Inputs: Use clear delimiters (e.g., backticks, brackets) to distinguish input text from instructions.

IV. Structured Output Requests: Specify expected structure (e.g., table, list, paragraph) to obtain organized responses.

For example, if asking for information on Golden Retrievers, a precise prompt like "Describe the behaviour and characteristics of Golden Retrievers in two paragraphs for a family audience" yields more targeted content than a vague request such as "Tell me about dogs."

3.3. Advanced Prompt Engineering Techniques

I. Zero-Shot Prompting: No example cited. The model responds based on prior training. Best for simple queries.

II. One-Shot Prompting: One example cited to establish response style or structure.

III. Few-Shot Prompting: Multiple examples cited. Useful for complex tasks such as sentiment analysis or entity classification (Datacamp, 2025).

Example:

prompt = """

Q: Given {1, 3, 7, 12, 19}, identify the odd numbers.

A: {1, 3, 7, 19}

Q: Given {3, 5, 11, 12, 16}, identify the odd numbers.

A:

"""

iv. Multi-Step Prompting

This technique breaks down a task into multiple steps. For example, planning a trip can be structured into four sequential prompts:

Suggest destinations.

List accommodations.

Recommend activities.

Compare pros and cons.

This results in more comprehensive and logically sequenced outputs.

V. Chain-of-Thought and Self-Consistency Prompting

Chain-of-thought prompting encourages the model to show reasoning before arriving at an answer. This is particularly helpful for math problems or logical analysis. Self-consistency involves multiple reasoning paths, selecting the most frequent response (majority vote) to ensure reliability.

VI. Conditional Prompts

Conditional logic can guide responses based on input. For instance: "If the input text is in English and contains the keyword 'technology', generate a title. Else, respond with 'Keyword not found'."

3.4. Iterative Prompt Engineering and Refinement

Prompt engineering is iterative. Initial outputs may lack detail, be ambiguous, or misunderstand instructions. Refinement involves tweaking prompts for clarity, specificity, or structure. For example, if the model returns generic data, refining the prompt to request output in table format or include specific fields like "Release Year" can produce more actionable results.

This process applies to all prompt types. In few-shot prompts, refining the examples can guide better classification. In multi-step and chain-of-thought prompts, clearer step definitions or examples help the model perform more accurately (DataCamp, 2025).

3.5. Structured Output Techniques

When expecting outputs in structured formats:

Tables: Define headers (e.g., "Title", "Author", "Year").

Lists: Clarify whether the list should be ordered or unordered.

Paragraphs with Subheadings: Mention this explicitly.

Custom Formats: Combine formatting instructions, output expectations, and input content clearly.

3.6. Prompt Components in RAG Architecture

Key components include:

- PromptTemplate: Structured input format with placeholders.
- conversation_messages: Maintains dialogue history.
- Prompt: User’s query, directing the LLM.

These are critical in Retrieval-Augmented Generation (RAG), where prompts interact with vector databases to fetch context before responding.

3.7. Evaluation Techniques for RAG-Based Outputs

Effective evaluation ensures model outputs are accurate and grounded in retrieved content. Common methods include:

Human Evaluation: Domain experts rate outputs based on quality.

BLEU/ROUGE: Measures text similarity with reference answers.

Faithfulness Score: Assesses if answers reflect source documents.

Semantic Similarity: Uses embeddings to gauge relevance.

Factuality Checks: Validates correctness (DataCamp, 2025).

For credit scoring applications, a custom rubric is best. Key criteria include:

- Relevance
- Data accuracy (e.g., Debt-to-Income values)
- Reasoning quality
- Clarity
- Grounding in retrieved documents.

3.8 Prompts & Large Language Models (LLMs)

Prompt engineering is essential for leveraging the full capabilities of LLMs. Through clear instruction, structured formatting, iterative refinement, and advanced prompting techniques such as few-shot or chain-of-thought, models can generate more accurate, relevant, and interpretable outputs. When combined with retrieval systems in RAG, the power of prompt engineering becomes even more critical, requiring thoughtful design and rigorous evaluation to ensure robust performance in high-stakes domains like finance.

Chapter 4

4. Methodology & Approach

4.1. Proof of Concept (POC) Development and Workflow

To reinforce and validate the core ideas underpinning this dissertation, a Proof of Concept (POC) was first researched & understudied using different Datacamp skill tracks considering gaps identified from research papers. This was then followed by a POC design and then implementation. The goal of the POC is to demonstrate the feasibility and effectiveness of using a Retrieval-Augmented Generation (RAG) architecture in conjunction with prompt engineering techniques for optimizing insights gathered from LLMs as it relates to credit score analysis. The entire workflow is outlined below:

- Data Selection: A credit score dataset was carefully selected to serve as the foundation for analysis and model interaction.
- Exploratory Data Analysis (EDA): Initial data inspection and analysis were performed to assess quality, detect patterns, and prepare the dataset for downstream processing.
- Generation of Key Business Questions: Business-relevant questions were formulated from the dataset to function as prompts for querying the language model.
- Retrieval-Augmented Generation (RAG): This phase involved document loading, text splitting, and vector embedding of the dataset using a selected vector database for efficient information retrieval.
- Integration of RAG Components with the LLM: The vector database and processed data were integrated with a Large Language Model to support intelligent, context-aware responses.
- LLM Querying Without Prompt Optimization: Initial prompts were evaluated without optimization to establish baseline performance.
- LLM Querying with Prompt Optimization: Optimized prompt engineering techniques were applied to improve response relevance, clarity, and accuracy.
- Response Evaluation and POC Validation: The outputs from both prompt styles were evaluated using appropriate metrics to validate the success of the proof of concept.

This structured workflow ensured that each stage of the project contributed meaningfully to demonstrating the viability of enhanced credit score analysis through prompt-optimized RAG-based LLM systems.

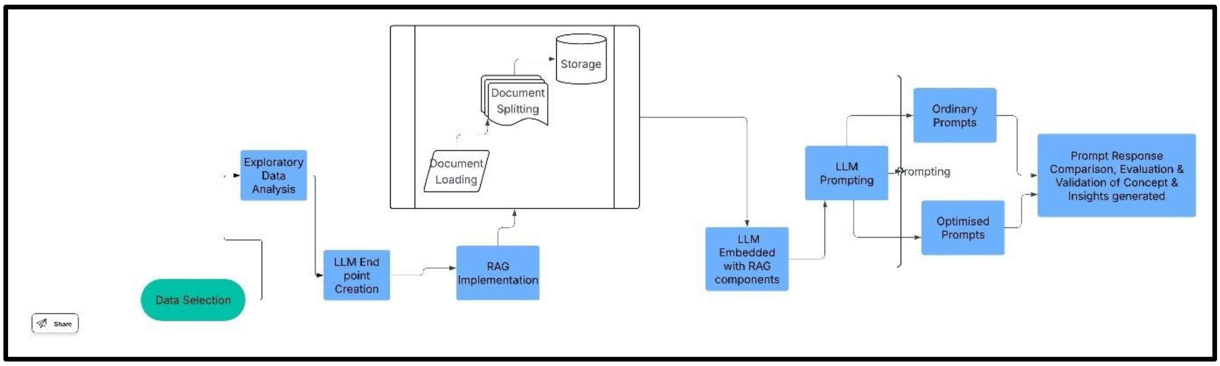

Figure 3: Flowchart showing Methodology Workflow

4.2. Data Selection

The dataset used for this project is a credit score dataset sourced from Kaggle, and it presents a rich and multifaceted collection of structured financial data designed to support machine learning and analytical tasks in credit risk evaluation.

4.3. Exploratory Data Analysis

Exploratory Data Analysis (EDA) served as a critical foundation for understanding and preparing the credit score dataset collected from Kaggle. The process commenced with data loading and structural inspection, which involved examining the dataset's shape, identifying the column names, data types, and assessing the number of records. This initial step provided a comprehensive overview of the dataset's composition and guided subsequent analysis.

Descriptive statistics were then generated for key numerical features such as INCOME, CREDIT_SCORE, and R_DEBT_INCOME to evaluate central tendencies, dispersion, and the presence of outliers. Checks for uniqueness and duplicate records were also performed to ensure data integrity and manage biased results.

To further examine the dataset, visual exploration techniques were employed. Histograms were used to assess the distribution and skewness of continuous variables like CREDIT_SCORE, while bar charts were used to evaluate the frequency distribution of categorical variables such as CAT_GAMBLE. These visualizations provided critical insights into potential imbalances and highlighted patterns in the data.

A feature importance analysis was also conducted to determine the relative contribution of each attribute to the CREDIT_SCORE variable. To enable this, categorical features were encoded into numerical values using appropriate transformation techniques, ensuring compatibility with machine learning algorithms.

Overall, the EDA phase was instrumental in identifying data quality issues, uncovering actionable insights, and informing the selection and engineering of features for downstream modeling. This step was especially essential given the integration with Retrieval-Augmented Generation (RAG)-based Large Language Models (LLMs), where contextual accuracy, feature consistency, and data reliability are paramount for generating credible and interpretable outputs.

4.4. Prompts Questions Generation

To develop effective prompts for a Retrieval-Augmented Generation (RAG) based Large Language Model (LLM) augmented with credit score data, it was essential to research the key insights typically derived by lenders, financial institutions, and businesses from credit scoring datasets. These datasets contain critical financial, demographic, and behavioural variables that influence an individual's creditworthiness and overall financial health. Insights into creditworthiness patterns emerge by identifying key attributes like payment history, debt-to-income ratio, credit utilization, and the length of credit history. These help in understanding the direct impact of missed payments or defaults on a borrower's score.

The features embedded within the credit score dataset serve as critical tools leveraged by lenders and financial institutions to evaluate customer behavior and determine eligibility for loan facilities. Attributes such as employment status, monthly income, loan amount requested, and repayment history are instrumental in identifying the key factors that influence loan approval decisions. These variables help in constructing detailed and accurate risk profiles of applicants. Moreover, the dataset empowers institutions to perform risk segmentation by classifying borrowers into low, medium, or high-risk categories based on historical financial behavior. It also aids in fraud detection by flagging anomalies such as repeated loan applications or inconsistencies in reported income, which may suggest suspicious activity.These insights form the backbone of the prompt structures used to engage the RAG-augmented LLM.

Below are potential questions considered for Prompts:

A. Credit Score & Risk Segmentation

- What is the individual's current credit score, and how has it changed over time?
- How does the individual's credit score compare to the average credit score in their demographic group?
- What risk category does the individual belong to (low-risk, medium-risk, or high-risk borrower)?

B. Payment Behaviour & Credit History

- What percentage of on-time vs. late payments has the individual made in the last 12 months?
- How many missed or delinquent payments has the individual had in the last 6–24 months?
- How long has the individual maintained a credit history (credit age)?
- Has the individual ever had an account sent to collections or declared bankruptcy?

C. Debt Management & Utilization

- What is the individual's credit utilization ratio (credit used vs. available credit)?
- What is the total outstanding debt across all loans and credit lines?
- How much of the individual’s monthly income is allocated to debt repayment (Debt-to-Income Ratio - DTI)?
- Does the individual have a history of maxing out credit cards or revolving high credit balances?

D. Loan & Credit Account Activity

- How many open credit accounts does the individual currently have?
- What types of credit does the individual have (credit cards, personal loans, mortgages, auto loans, etc.)?
- Has the individual recently applied for multiple loans or credit cards (number of hard inquiries in the last 6 months)?
- Are there high-risk loans (e.g., payday loans) in the individual’s credit portfolio?

E. Default & Fraud Risk Indicators

- Has the individual ever defaulted on a loan or had a loan restructuring due to financial hardship?

- Are there any suspicious financial activities, such as frequent large withdrawals or deposits?

- Has the individual’s spending behaviour changed suddenly and significantly (e.g., excessive gambling transactions)?

F. Economic & Demographic Insights

- How does the individual's financial behaviour compare to that of others in a similar income bracket?

- Is there a correlation between the individual's employment status and their ability to make payments on time?

4.5. Retrieval Augmented Generation

Pre-trained language models don't have access to external data sources - their understanding comes purely from their training data. This means that if we require our model to have knowledge that goes beyond its training data, which could be company data or knowledge of more recent world events, we need a way of integrating that data. In RAG, a user query is embedded and used to retrieve the most relevant documents from the database. Then, these documents are added to the model's prompt so that the model has extra context to inform its response.

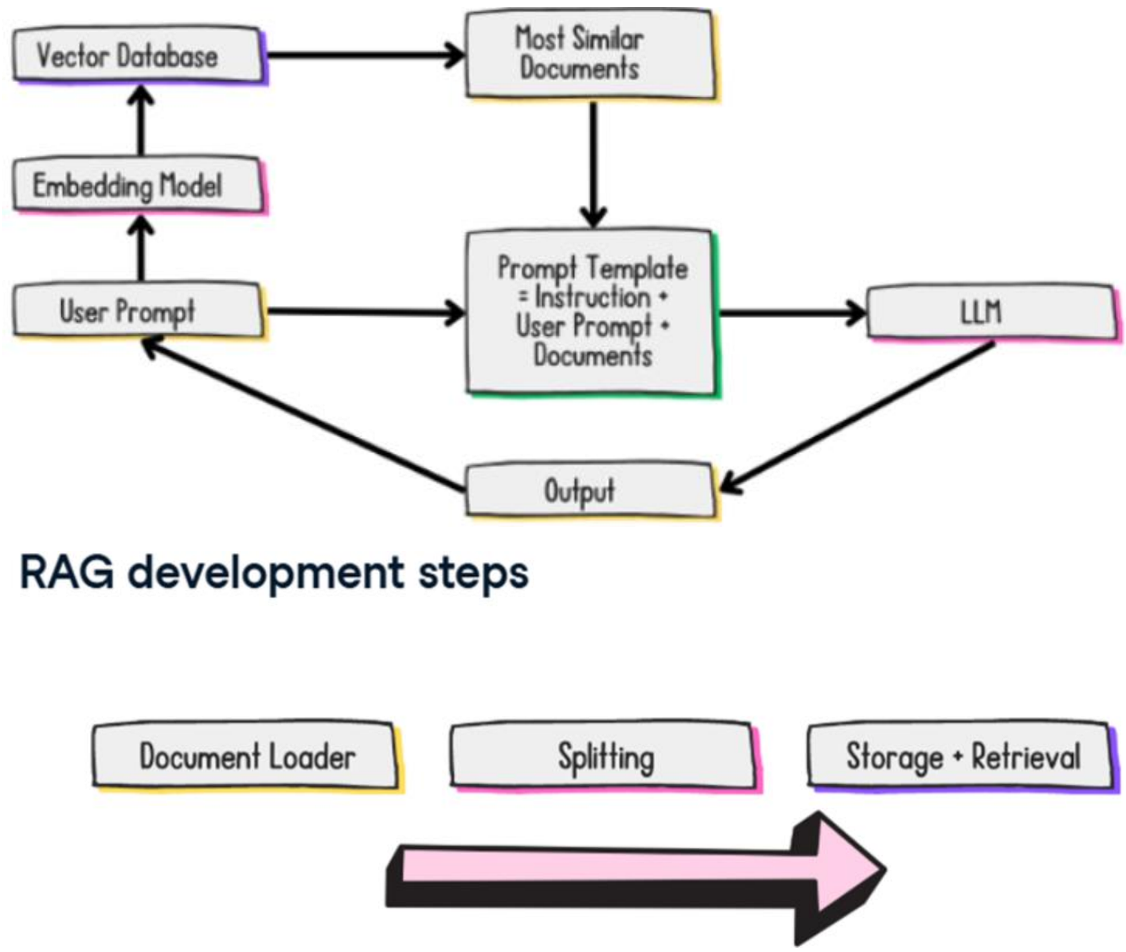

Figure 4: Flowchart showing workflow for RAG implementation (Datacamp, 2025)

4.6.RAG Implementation

The dataset was loaded using the Pandas library since it is a structured dataset. Since the document is structured data, it was loaded as a simple dataframe. Code snippet

```
#Load data of interest

df = pd.read_excel("credit_score.xlsx")

#Checking first 5 rows

df.head()
```

Document Splitting Strategy

Row-wise (Per-Customer) Splitting was applied since the credit score is a structured data where each row represents a single customer's data. This allowed each chunk to contain the entire row for a specific customer_id. This enabled features for a customer to stay together in one chunk and the LLM can reference complete customer data during a prompt.

4.7. Integrate RAG components with LLM

Two LLM versions (GPT-4o-mini & GPT-4o) of the OpenAI large language Model were initially tested and the better model of the two was selected for the RAG pipeline implementation. The ChatOpenAI class from the langchain_openai partner library was used to make a request to the OpenAI API for response. To access the OpenAI API, a mandatory API key was incurred, and this was stored in a .env file and accessed via the load_dotenv() library. The ChatOpenAI class accepts parameters like max_completion_tokens and temperature. To prompt this model, the .invoke() method was called on a prompt string.

The RAG development process was initiated by a document loader step using a simple pandas method since the data is a structured dataset.

The row-wise document splitting strategy was adopted knowing it is a structured dataset, and this permitted all features for each customer to stay together in one chunk while allowing the LLM to reference complete customer data during a prompt.

Two vector databases were evaluated for performance, with the more effective option selected for further use. The first database evaluated was **FAISS**, and the second was the **Weaviate** vector database. A dedicated function was developed to integrate key argument components and facilitate interaction with the chosen Large Language Model (LLM).

Additionally, custom pipelines were encapsulated within modular functions to streamline the communication of prompts to the LLM. Each function was designed to manage specific arguments, ensuring optimal compatibility and performance in response generation.

4.7b. Prompt Optimization Strategy

For optimal prompt customization, three separate prompt arguments were designed. These arguments include:

(a) Custom Prompt Template
(b) Prompt
(c) Conversation Prompt

(a) Custom Prompt Template

A Custom Prompt Template is a pre-defined structure or format used to generate dynamic prompts for LLMs. It typically contains placeholders or variables (e.g. question}, {context}, {chat_history}) that are filled in at runtime based on user input, retrieved documents, or prior interactions. Custom prompt templates help standardize how instructions and context are provided to an LLM, improving consistency, accuracy, and clarity of the model's responses.

Example:

Template: "Given the customer credit data: {context}, provide an analysis for the question: {question}" (OpenAI, 2025)

(b) Prompt

A Prompt is the input or instruction given to a language model that tells it what to do. It can range from a simple question to a complex multi-part request. Prompts guide the model's output and define the task scope, format, and expected behavior. Effective prompting is critical for accurate and relevant model responses, especially when dealing with domain-specific tasks like credit scoring, summarization, or reasoning.

Example:

Prompt: "What is the customer's credit score based on the following parameters: income, debt ratio, and loan history?" (OpenAI, 2025).

(c) **Conversation Prompt**

A Conversation Prompt refers to a structured history of messages exchanged between a user and an LLM in a chat-like format. It typically includes alternating roles such as "user", "assistant", and optionally "system" messages. Conversation prompts are especially important in multiturn interactions where context from previous exchanges must be retained for coherence and continuity.

Example of a conversation prompt structure:

```
[
  {"role": "system", "content": "You are a helpful credit scoring assistant."},
```

{"role": "user", "content": "Can you explain credit utilization?"},

{"role": "assistant", "content": "Credit utilization refers to the ratio of used credit to available credit."}] (OpenAI, 2025).

These arguments combined had a major goal of shaping and customizing prompt for optimization.

4. 8. Embedded LLM Querying without Prompt Optimization

All arguments for the above highlighted functions were applied in their default state.

The document argument was confirmed as shown below.

Input

#View the created document

print(documents[0].page_content[:50])

Output

```
Customer ID: C02COQEVYU
CUST_ID: C02COQEVYU
INCOME
```

The prompt template argument was confirmed as shown below.

```
from langchain.prompts import PromptTemplate


default_custom_prompt_style = PromptTemplate(
    input_variables=["question", "chat_history"],
    template="""
Given the following conversation and a follow up question, rephrase the follow up question to be a standalone question in two sentences at most.
Chat History:
{chat_history}


Follow Up Question: {question}


Standalone Question:
"""
)
```

The conversation messages argument was confirmed as shown below.

```
default_conversation_messages = [
    ("Kindly respond to the Question", "")
]
```

4.9. Prompt Test 1

```
#prompts
prompt1 = """
How does this customer's (CZRA4MLB0P) credit score compare to the average credit score (586.7) and what debt category does the individual belong to?

 """
conversation_messages = [(prompt1, "")]  # Empty assistant response for new conversation
```

LLM components put together & Function Called

Rag Based LLM Response : Prompt 1 Not Optimized

```
response = query_credit_score_bot_weaviate(
    custom_qa_prompt=default_custom_prompt_style,
    prompt_question=prompt1,
    conversation_messages=default_conversation_messages
)
```

4.10. Prompt Test 2 (Not Optimized)

```
#prompts
prompt2 = """
For Customer ID CZRA4MLB0P What is the total outstanding debt across all loans and credit lines and
How much of the individual’s monthly income is allocated to debt repayment (Debt-to-Income Ratio - DTI)?


 """
```

LLM components put together & Function Called

Rag Based LLM Response : Prompt 2 Not Optimized

```
response_2 = query_credit_score_bot_weaviate (
    custom_qa_prompt=default_custom_prompt_style,
    prompt_question=prompt2,
    conversation_messages=default_conversation_messages
)
```

4.11. Prompt Test 3 (Not Optimized)

```
#prompts
prompt3 = """
What are the travel expenses or inflow that affect this customer with Cust ID CZRA4MLB0P and How does this affect this customer's credit score?


 """
conversation_messages = [(prompt3, "")] # Empty assistant response for new conversation
```

LLM components put together & Function Called

```
response_3 = query_credit_score_bot_weaviate (
    custom_qa_prompt=default_custom_prompt_style,
    prompt_question=prompt3,
    conversation_messages=default_conversation_messages
)
```

4.12. Embedded LLM Querying with Prompt Optimization

All prompts initially passed were afterwards optimized with selected prompt techniques.

The code snippets and prompt customization of each argument is outlined below.

4.13. First Optimized Prompt

```
prompt1_Op = (
    "Compare the credit score of Customer ID CZRA4MLB0P with the overall average credit score from the dataset. "
```

"Based on this, classify the customer debt category: Low, Medium, or High."

)

Prompt Optimization strategy here was based on the following considerations :

- Emphasis on clarity
- Comparative Prompting
- Risk Indicator Prompting

This combination works best because the goal is to compare an individual's credit score to an average ("Compare") with a clear request to identify the risk category ("Low", "Medium", or "High").

The conversation message used.

conversation_messages1_op = [

("""The dataset includes credit scores, from these credit score average credit score of 586.7 can be deduced. Also

the dataset has a debt column that shows the debt owed by each customer, look into the debt column to deduce what category this customer falls""", "")

]

The Prompt Template used.

from langchain.prompts import PromptTemplate

custom_prompt_style1_op = PromptTemplate (

input_variables= ["question", "chat_history"],

template="""

You are a helpful credit data assistant. Using the available dataset, answer the user's question thoroughly.

The dataset includes fields like INCOME, DEBT, SAVINGS, CREDIT_SCORE, and other financial metrics.

Respond clearly using insights from similar records. If comparisons or averages are implied, include them.

Chat History:

{chat_history}

Follow Up Question: {question}

Standalone Question:
"""
)

LLM components put together & Function Called

```
response_op = query_credit_score_bot_weaviate (
    custom_qa_prompt=custom_prompt_style1_op,
    prompt_question=prompt1_Op,
    conversation_messages=conversation_messages1_op
)
```

4.14. Second Optimized Prompt

Multi-style prompting was employed in this section as an advanced prompting technique to enhance the quality and interpretability of responses. This method builds upon and extends the concept of labelled guide prompting proposed by Teixeira et al. (2023), which focuses primarily on labeling rather than providing instructive or stepwise guidance. Given the sensitive nature of financial applications, multi-style prompting was chosen for its strengths in improving interpretability, enhancing response quality, and mitigating hallucinations—critical factors for ensuring reliability and transparency in credit score insight generation.

Second Optimized Prompt Using Multi-Style Prompting Technique

```
prompt2_Op = (
    "Step 1: Retrieve the total outstanding DEBT for Customer ID CZRA4MLB0P.\n”
    "Step 2: Retrieve the monthly INCOME for the same customer.\n”
    "Step 3: Calculate the Debt-to-Income Ratio (DTI = DEBT / INCOME).\n"
    "Step 4: Provide the percentage of the individual’s monthly income allocated to debt repayment.\n"
    "Step 5: Provide an interpretation of the DTI score.\n"


)
```

Note : Multi-Step Prompt LLM was instructed to follow an ordered reasoning path, which aligns with multi-step prompting.

The Conversation Messages used

```
conversation_messages2_op = [
    (
        "We're analysing customer financials. Use dataset fields like DEBT, INCOME, and SAVINGS to compute useful metrics like Debt-to-Income (DTI). Start by pulling these values for CZRA4MLB0P.",
        "Understood. I'll extract DEBT and INCOME from the dataset, calculate DTI, and interpret the result."
    )
]
```

Prompt_template_Used

```
from langchain.prompts import PromptTemplate


custom_prompt_style2_op = PromptTemplate(
    input_variables= ["question", "chat_history"],
    template="""
You are a financial assistant skilled in analysing customer credit data.


Your task is to respond clearly using multi-step reasoning. The dataset includes DEBT, INCOME, CREDIT_SCORE, SAVINGS, and related fields. Use these to extract values, compute ratios, and provide clear conclusions.


Chat History:
{chat_history}


Follow Up Question: {question}


Answer in 3–5 sentences with a breakdown of your reasoning.
"""
)
```

LLM components put together & Function Called

```
response2_op = query_credit_score_bot_weaviate (
    custom_qa_prompt=custom_prompt_style2_op,
    prompt_question=prompt2_Op,
    conversation_messages=conversation_messages2_op
)
```

4.15. Third Optimized Prompt

Chain-of-thought prompting was utilized in this section as an advanced technique to improve the transparency, traceability, and interpretability of responses. This approach is particularly effective in managing complex decision-making scenarios that require consideration of multiple factors. Additionally, it plays a critical role in mitigating hallucinations by guiding the LLM to maintain logical, step-by-step reasoning grounded in retrieved source documents, thereby reducing the likelihood of generating unsupported or fabricated content.

Third Optimized Prompt implemented using the Chain of thought prompt.

```
prompt3_op = """
Analyse the travel-related expenses and income for the customer with Cust ID CZRA4MLB0P.
First, identify if the customer has any notable travel-related DEBT, SAVINGS, or INCOME records.
Then, step-by-step, reason how each factor (debt, savings, income) could potentially affect the customer's credit score.
Finally, conclude with a brief evaluation of the overall credit impact for this customer.
"""
```

Selected Prompt Type:

Chain of thought prompt

Chain-of-thought prompting requires language models to present reasoning steps, or thoughts, before giving a final answer. This technique is valuable for complex reasoning tasks and helps reduce errors by processing the reasoning step by step.

Conversation message used.

```
conversation_messages3_op = [
```

{"role": "system", "content": """The dataset contains fields such as CREDIT_SCORE, INCOME, DEBT, SAVINGS, and others.

The average CREDIT_SCORE across the dataset is approximately 586.7.

Important context:

- DEBT column shows total outstanding debts (including travel-related debts).

- INCOME may include travel income if relevant.

- SAVINGS indicates financial discipline, including travel savings.

Use this context to first find where the customer falls in debt or income categories compared to others,

and reason about the intermediate factors before concluding the effect on credit score. Give a concise response"""}

]

Prompt Template Used

```
from langchain.prompts import PromptTemplate

custom_prompt_style3_op = PromptTemplate(
    input_variables= ["question", "chat_history"],
    template="""
You are a financial credit analyst assistant specialized in customer behaviour.

Your task is to assist the user in analysing financial datasets to deduce credit score impacts.

Ensure your responses are concise without unnecessary information.

The dataset includes columns like INCOME, DEBT, SAVINGS, CREDIT_SCORE, and others.

**Important Instructions:**
1. Break down your reasoning step-by-step (Chain-of-Thought).
2. First, identify the relevant data fields.
3. Compare or benchmark where necessary (e.g., against averages).
4. Explain intermediate deductions logically.
5. Finally, summarize the credit score implications clearly.

Use examples from similar customers if it helps.
---
Chat History:
{chat_history}
```

```
Follow-Up Question:
{question}

Standalone Version of the Question:
"""
)
```

LLM components put together & Function Called

```
response3_op = query_credit_score_bot_weaviate (
    custom_qa_prompt=custom_prompt_style3_op,
    prompt_question=prompt3_op,
    conversation_messages=conversation_messages3_op
)
```

4.16. Evaluation of Output

Evaluation of output was executed using both qualitative and quantitative approach. A comparison of the RAG based LLM outputs for both standard/ordinary prompts and optimized prompts were analysed using the following evaluation metrics:

Qualitative Evaluation using ChatGPT

- Relevance to Prompt
- Data Accuracy
- Reasoning Quality
- Faithfulness to source Documents
- Clarity & Structure

Quantitative Evaluation

Bilingual Evaluation Understudy (BLEU)

BLEU Measures precision by stating how many words or phrases in the generated response appear in the reference. Often used in machine translation (OpenAI, 2025)

To evaluate the performance of the RAG-based LLM, a reference output was constructed by carefully analysing the credit score dataset and defining the expected response for each prompt. The model's responses generated using both unoptimized and optimized prompts were subsequently assessed. Prior to applying the BLEU (Bilingual Evaluation Understudy) metric, each response was pre-processed by removing extraneous phrases and non-essential words while preserving the core semantic content relevant to the original question. This approach ensured a fair and focused comparison of the generated outputs against the reference, allowing for a more accurate measurement of precision and content alignment using BLEU

Chapter 5

5.Result & Findings

5.1. Findings During Exploratory Data Analysis

The Kaggle credit scoring dataset demonstrated high data quality, with no duplicate entries or missing values across all columns. A total of 259 unique credit score records were observed. Preliminary statistical analysis of the credit score variable revealed a slight left skew in its distribution, as indicated by a skewness value of -1.005. Additionally, the kurtosis value of 2.03 suggests a distribution that is slightly flatter than a normal distribution. These insights are essential for understanding the underlying data characteristics and ensuring appropriate preprocessing steps for downstream tasks such as modeling and integration with the Retrieval-Augmented Generation (RAG) system.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Load data of inetrest
df = pd.read_excel("credit_score.xlsx")
#Checking first 5 rows
df.head()
```

	CUST_ID	INCOME	SAVINGS	DEBT	R_SAVINGS_INCOME	R_DEBT_INCOME	R_DEBT_SAVINGS	T_CLOTHING_12	T_CLOTHING_6	R_CLOTHING	...	R_EXPENDITU
0	C02COQEVYU	33269	0	532304	0.0000	16.0000	1.2000	1889	945	0.5003	...	
1	C02OZKC0ZF	77158	91187	315648	1.1818	4.0909	3.4615	5818	111	0.0191	...	
2	C03FHP2D0A	30917	21642	534864	0.7000	17.3000	24.7142	1157	860	0.7433	...	
3	C03PVPPHOY	80657	64526	629125	0.8000	7.8000	9.7499	6857	3686	0.5376	...	
4	C04J69MUX0	149971	1172498	2399531	7.8182	16.0000	2.0465	1978	322	0.1628	...	

Figure 5a: Dataset Loaded and previewed in Notebook environment for EDA

```
#checking for data types and missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 87 columns):
 #    Column                    Non-Null Count   Dtype
---   ------                    --------------   -----
 0    CUST_ID                   1000 non-null    object
 1    INCOME                    1000 non-null    int64
 2    SAVINGS                   1000 non-null    int64
 3    DEBT                      1000 non-null    int64
 4    R_SAVINGS_INCOME          1000 non-null    float64
 5    R_DEBT_INCOME             1000 non-null    float64
 6    R_DEBT_SAVINGS            1000 non-null    float64
 7    T_CLOTHING_12             1000 non-null    int64
 8    T_CLOTHING_6              1000 non-null    int64
 9    R_CLOTHING                1000 non-null    float64
 10   R_CLOTHING_INCOME         1000 non-null    float64
```

Figure 5b: Snapshot of the data inspection carried out on Credit score dataset showing no null values

```
df.describe()
```

	INCOME	SAVINGS	DEBT	R_SAVINGS_INCOME	R_DEBT_INCOME	R_DEBT_SAVINGS	T_CLOTHING_12	T_CLOTHING_6	R_CLOTHING	R_CLOTHING
count	1000.000000	1.000000e+03	1.000000e+03	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	10
mean	121610.019000	4.131896e+05	7.907180e+05	4.063477	6.068449	5.867252	6822.401000	3466.320000	0.454848	
std	113716.699591	4.429160e+05	9.817904e+05	3.968097	5.847878	16.788356	7486.225932	5118.942977	0.236036	
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	30450.250000	5.971975e+04	5.396675e+04	1.000000	1.454500	0.206200	1084.500000	319.500000	0.263950	
50%	85090.000000	2.738505e+05	3.950955e+05	2.545450	4.911550	2.000000	4494.000000	1304.000000	0.468850	
75%	181217.500000	6.222600e+05	1.193230e+06	6.307100	8.587475	4.509600	10148.500000	4555.500000	0.626300	
max	662094.000000	2.911863e+06	5.968620e+06	16.111200	37.000600	292.842100	43255.000000	39918.000000	1.058300	

Figure 6: Statistical summary of credit score dataset

```
df.nunique()
```

```
CUST_ID                1000
INCOME                  951
SAVINGS                 992
DEBT                    943
R_SAVINGS_INCOME        463
                       ...
CAT_MORTGAGE              2
CAT_SAVINGS_ACCOUNT       2
CAT_DEPENDENTS            2
CREDIT_SCORE            259
DEFAULT                   2
Length: 87, dtype: int64
```

Figure 7: Check for uniqueness with 259 unique credit score values

[14]:
```python
#Check distribution for credit score column
plt.figure(figsize=(8,5))
sns.countplot(x='CREDIT_SCORE', data=df, palette='viridis', hue='CREDIT_SCORE', legend = True)
plt.title('Distribution of Credit Scores')
plt.show()
```

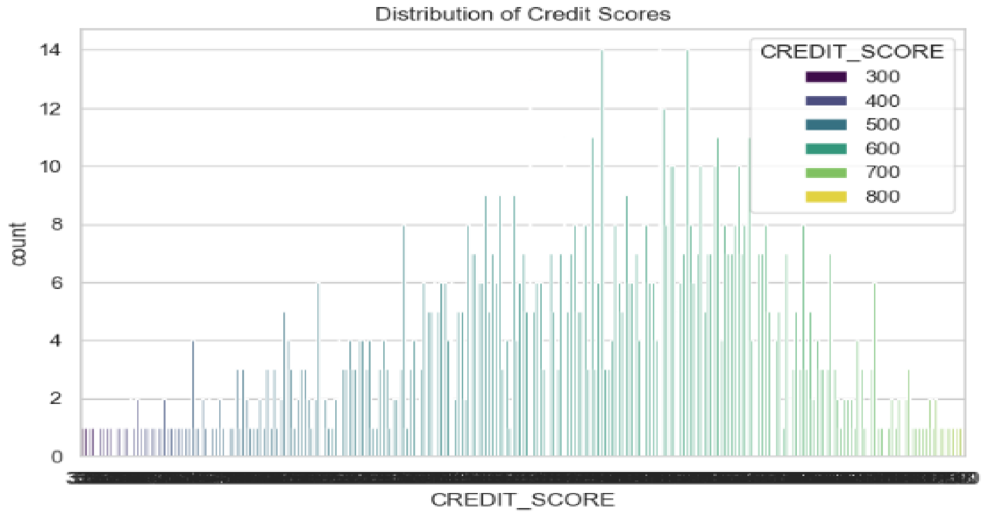

Figure 8: Observing the distribution of the credit score variable showing slight left skewness

[18]:
```python
#Feature Importance Analysis

#Encoding of categorical features
encoder = OrdinalEncoder()
df1['CODE_CAT_GAMBLING_'] = encoder.fit_transform(df1[['CAT_GAMBLING']])
df1['CODE_CUST_ID'] = encoder.fit_transform(df1[['CUST_ID']])
df1.head()
```

[18]:

	CUST_ID	INCOME	SAVINGS	DEBT	R_SAVINGS_INCOME	R_DEBT_INCOME	R_DEBT_SAVINGS	T_CLOTHING_12	T_CLOTHING_6	R_CLOTHING	...	CAT_GAMBLI
0	C02COQEVYU	33269	0	532304	0.0000	16.0000	1.2000	1889	945	0.5003	...	H
1	C02OZKC0ZF	77158	91187	315648	1.1818	4.0909	3.4615	5818	111	0.0191	...	
2	C03FHP2D0A	30917	21642	534864	0.7000	17.3000	24.7142	1157	860	0.7433	...	H
3	C03PVPPHOY	80657	64526	629125	0.8000	7.8000	9.7499	6857	3686	0.5376	...	H
4	C04J69MUX0	149971	1172498	2399531	7.8182	16.0000	2.0465	1978	322	0.1628	...	H

5 rows × 89 columns

Figure 9: Encoding of categorical variables to check to carry out feature importance analysis

```
[19]: #Feature Importance Analysis
#To determine the features that affect the credit score in a structured dataset
#Mutual Information: Measures the dependency between features and credit score.
from sklearn.feature_selection import mutual_info_regression

# Exclude specific columns
exclude = ['CUST_ID', 'CAT_GAMBLING','CREDIT_SCORE']

# Select numerical columns, excluding categorical and unwanted columns
numerical_columns = df1.select_dtypes(include=['number']).columns
filtered_columns = [col for col in numerical_columns if col not in exclude]

X = df1[filtered_columns]  # Exclude target variable
y = df1['CREDIT_SCORE']
mi_scores = mutual_info_regression(X, y)

feature_importance = pd.Series(mi_scores, index=X.columns).sort_values(ascending=False)
print(feature_importance.head(50))
```

```
R_DEBT_INCOME             0.633587
R_EXPENDITURE_DEBT        0.404516
R_UTILITIES_DEBT          0.342391
R_ENTERTAINMENT_DEBT      0.320156
R_TAX_DEBT                0.307736
R_DEBT_SAVINGS            0.288994
R_CLOTHING_DEBT           0.269767
R_GROCERIES_DEBT          0.258132
R_HEALTH_DEBT             0.251509
R_TRAVEL_DEBT             0.238933
```

Figure 10: Feature Importance Analysis of predictor variables with the Debt-to-Income ratio showing highest importance.

5.2. Prompts Questions Generation

Considering credit, risk and debt segmentation, the following questions were used as prompts

A. How does this customer's (CZRA4MLB0P) credit score compare to the average credit score (586.7) and what debt category the individual belongs to?

B. For Customer ID CZRA4MLB0P What is the total outstanding debt across all loans and credit lines and how much of the individual’s monthly income is allocated to debt repayment (Debt-to-Income Ratio - DTI)?

C. What are the travel expenses or inflow that affect this customer with Cust ID CZRA4MLB0P and How does this affect this customer's credit score?

5.3. Retrieval Augmented Generation Outputs

5.3A. RAG Implementation: Document Loading

The dataset was loaded using the Pandas library since it is a structured dataset. Since the document is structured data , it was loaded as a simple dataframe. Code snippet

Load data of interest

df = pd.read_excel("credit_score.xlsx")

#Checking first 5 rows

df.head()

	CUST_ID	INCOME	SAVINGS	DEBT	R_SAVINGS_INCOME	R_DEBT_INCOME	R_DEBT_SAVINGS	T_CLOTHING_12	T_CLOTHING_6	R_CLOTHING	...	R_EXPENDITU
0	C02COQEVYU	33269	0	532304	0.0000	16.0000	1.2000	1889	945	0.5003	...	
1	C02OZKC0ZF	77158	91187	315648	1.1818	4.0909	3.4615	5818	111	0.0191	...	
2	C03FHP2D0A	30917	21642	534864	0.7000	17.3000	24.7142	1157	860	0.7433	...	
3	C03PVPPHOY	80657	64526	629125	0.8000	7.8000	9.7499	6857	3686	0.5376	...	
4	C04J69MUX0	149971	1172498	2399531	7.8182	16.0000	2.0465	1978	322	0.1628	...	

5 rows × 87 columns

Figure 11: The first 5 rows loaded from the credit score dataset

Document Loader : The credit dataset will be loaded into the platform

```
#Alternative method to load data
from langchain_community.document_loaders import UnstructuredExcelLoader
loader = UnstructuredExcelLoader("credit_score.xlsx")
data = loader.load()

#Note : Since the document is a structured data , it can be loaded as a simple dataframe,  hence the inital loading "df" will be used.
# Load data of inetrest
df = pd.read_excel("credit_score.xlsx")
#Checking first 5 rows
df.head()
```

	CUST_ID	INCOME	SAVINGS	DEBT	R_SAVINGS_INCOME	R_DEBT_INCOME	R_DEBT_SAVINGS	T_CLOTHING_12	T_CLOTHING_6	R_CLOTHING	...	R_EXPENDIT
0	C02COQEVYU	33269	0	532304	0.0000	16.0000	1.2000	1889	945	0.5003	...	
1	C02OZKC0ZF	77158	91187	315648	1.1818	4.0909	3.4615	5818	111	0.0191	...	
2	C03FHP2D0A	30917	21642	534864	0.7000	17.3000	24.7142	1157	860	0.7433	...	
3	C03PVPPHOY	80657	64526	629125	0.8000	7.8000	9.7499	6857	3686	0.5376	...	
4	C04J69MUX0	149971	1172498	2399531	7.8182	16.0000	2.0465	1978	322	0.1628	...	

Figure 12: The first 5 rows loaded from the credit score dataset for document loading phase in RAG Implementation

5.3B. RAG Implementation: Document Splitting Strategy

The row-wise (per-customer) splitting applied allowed each chunk to contain the entire row for a specific customer_id. This enabled features for a customer to stay together in one chunk and the LLM can reference complete customer data during a prompt.

```
from langchain.schema import Document

# document to house all credit data info
documents = []
for _, row in df.iterrows():
    customer_id = row['CUST_ID']
    content = f"Customer ID: {customer_id}\n" + "\n".join([f"{col}: {val}" for col, val in row.items()])
    documents.append(Document(page_content=content, metadata={"CUST_ID": customer_id}))
```

```
#View the created document
print(documents[0].page_content[:100])
```

```
Customer ID: C02COQEVYU
CUST_ID: C02COQEVYU
INCOME: 33269
SAVINGS: 0
DEBT: 532304
R_SAVINGS_INCOME:
```

Figure 13: Splitting of credit score dataset by showing the first 100 characters of the first row.

5.3C. RAG Implementation: Document storage using vector databases

The Weaviate vector database displayed a better search algorithm compared to the FAISS vector database, hence the Weaviate vector database was used for the RAG-based LLM pipeline development .

```python
#Using GPT 4o-mini & Faiss Vector db
def lang_chain_credit_score_bot_faiss(custom_prompt_feed, prompt_question, documents, conversation_messages):
    # Dependencies
    from dotenv import load_dotenv
    import time
    from langchain_openai import OpenAIEmbeddings, ChatOpenAI
    from langchain_community.vectorstores import FAISS
    from langchain_core.documents import Document
    from langchain.prompts import PromptTemplate
    from langchain.chains import ConversationalRetrievalChain
    from langchain.memory import ConversationBufferMemory
    from langchain.schema import AIMessage, HumanMessage, SystemMessage

    # Load .env
    load_dotenv()

    # Initialize LLM
    llm_1 = ChatOpenAI(model="gpt-4o-mini")

    # Embeddings
    embedding_function = OpenAIEmbeddings(model="text-embedding-3-small")

    # Initialize FAISS vectorstore
    batch_size = 200
    vectorstore = None

    for i in range(0, len(documents), batch_size):
        batch = documents[i:i+batch_size]
        if vectorstore is None:
            vectorstore = FAISS.from_documents(batch, embedding_function)
        else:
            vectorstore.add_documents(batch)
        time.sleep(1)

    retriever = vectorstore.as_retriever(search_type="similarity", search_kwargs={"k": 3})

    # Convert conversation_messages to LangChain Message types
    formatted_messages = []
    for msg in conversation_messages:
        role = msg["role"]
        content = msg["content"]
        if role == "user":
            formatted_messages.append(HumanMessage(content=content))
        elif role == "assistant":
            formatted_messages.append(AIMessage(content=content))
        elif role == "system":
            formatted_messages.append(SystemMessage(content=content))

    # Create memory and preload with conversation history
    memory = ConversationBufferMemory(
        memory_key="chat_history"
```

Figure 14a: Document storage using FAISS vector database

```
#embeddings using weaviate vector db
def setup_credit_score_collection(documents):
    import os
    from dotenv import load_dotenv
    from weaviate import WeaviateClient
    from weaviate.connect import ConnectionParams
    from weaviate.classes.config import Property, DataType, Configure
    from langchain_openai import OpenAIEmbeddings
    from langchain.schema import Document
    from langchain_weaviate import WeaviateVectorStore

    load_dotenv()

    client = WeaviateClient(
        connection_params=ConnectionParams.from_params(
            http_host="localhost", http_port=8080, http_secure=False,
            grpc_host="localhost", grpc_port=50051, grpc_secure=False
        ),
        skip_init_checks=True
    )

    client.connect()

    if not client.collections.exists("CreditScore"):
        client.collections.create(
            name="CreditScore",
            properties=[
                Property(name="text", data_type=DataType.TEXT)
            ],
            vectorizer_config=Configure.Vectorizer.none()
        )

        print("✅ CreditScore collection created.")

    else:
        print("ℹ️ CreditScore collection already exists. Skipping setup.")

    # Upload documents whether or not the collection existed (you can add a flag if needed)
    embeddings = OpenAIEmbeddings(model="text-embedding-3-small")

    # 👉 Ensure that your document text is aligned with "text" field
    vectorstore = WeaviateVectorStore(
        client=client,
        index_name="CreditScore",
```

Figure 14b: Document storage using Weaviate vector database

```
[82]: #Test RAG system for GPT 4.0-mini using faiss vectore db
      lang_chain_credit_score_bot_faiss(custom_prompt_style, prompt, documents, conversation_messages_2)

      I don't know the credit score associated with Customer ID: CZRA4MLB0P.
```

```
setup_credit_score_collection_edited(documents)

ℹ️ CreditScore collection already exists. Skipping setup
✅ Uploaded 254 documents in a batch.
✅ Uploaded 254 documents in a batch.
✅ Uploaded 254 documents in a batch.
✅ Uploaded final batch with 238 documents.
🎯 Total uploaded documents: 1000
```

Figure 14c: Comparing output of the FAISS and Weaviate vector database with Weaviate having a better performance.

5.4. Integration of RAG components with LLM

In a bid, identify possible cost and performance maximization alongside the trade-offs that could be explored, two LLM versions (GPT-4o-mini & GPT-4o) of the OpenAI large language Model were initially evaluated and the better model of the two was selected for the RAG pipeline implementation. The ChatOpenAI class from the langchain_openai partner library was used to make a request to the OpenAI API for response. To access the OpenAI API, a mandatory API key was incurred, and this was stored in a .env file and accessed via the load_dotenv() library. The ChatOpenAI class accepts parameters like max_completion_tokens and temperature. To prompt this model, the .invoke() method was called on a prompt string.

A dedicated function was developed to integrate key argument components and facilitate interaction with the chosen Large Language Model (LLM).

Additionally, custom pipelines were encapsulated within modular functions to streamline the communication of prompts to the LLM. Each function was designed to manage specific arguments, ensuring optimal compatibility and performance in response generation.

```
#Using GPT 4o-mini & Faiss Vector db
def lang_chain_credit_score_bot_faiss(custom_prompt_feed, prompt_question, documents, conversation_messages):
    # Dependencies
    from dotenv import load_dotenv
    import time
    from langchain_openai import OpenAIEmbeddings, ChatOpenAI
    from langchain_community.vectorstores import FAISS
    from langchain_core.documents import Document
    from langchain.prompts import PromptTemplate
    from langchain.chains import ConversationalRetrievalChain
    from langchain.memory import ConversationBufferMemory
    from langchain.schema import AIMessage, HumanMessage, SystemMessage

    # Load .env
    load_dotenv()

    # Initialize LLM
    llm_1 = ChatOpenAI(model="gpt-4o-mini")

    # Embeddings
    embedding_function = OpenAIEmbeddings(model="text-embedding-3-small")

    # Initialize FAISS vectorstore
    batch_size = 500
    vectorstore = None

    for i in range(0, len(documents), batch_size):
        batch = documents[i:i+batch_size]
        if vectorstore is None:
            vectorstore = FAISS.from_documents(batch, embedding_function)
        else:
```

```
            vectorstore = FAISS.from_documents(batch, embedding_function)
        else:
            vectorstore.add_documents(batch)
        time.sleep(1)

    retriever = vectorstore.as_retriever(search_type="similarity", search_kwargs={"k": 3})

    # Convert conversation_messages to LangChain Message types
    formatted_messages = []
    for msg in conversation_messages:
        role = msg["role"]
        content = msg["content"]
        if role == "user":
            formatted_messages.append(HumanMessage(content=content))
        elif role == "assistant":
            formatted_messages.append(AIMessage(content=content))
        elif role == "system":
            formatted_messages.append(SystemMessage(content=content))

    # Create memory and preload with conversation history
    memory = ConversationBufferMemory(
        memory_key="chat_history",
        return_messages=True
    )
    memory.chat_memory.messages = formatted_messages  # Inject messages

    # Build chain
    qa_chain = ConversationalRetrievalChain.from_llm(
        llm=llm_1,
        retriever=retriever,
        memory=memory,
        condense_question_prompt=custom_prompt_feed
    )

    # Invoke question
    query = prompt_question
    response = qa_chain.invoke({
    "question": query
   # "context": ""  # Or some string/summary if your prompt requires it
})
    print(response["answer"])
```

Figure 15: First Function using GPT 4.0-mini and four arguments FAISS vector database

```
#using GPT 4.o
def lang_chain_credit_score_bot4o_faiss(custom_prompt_feed, prompt_question, documents, conversation_messages):
    # Dependencies
    from dotenv import load_dotenv
    import time
    from langchain_openai import OpenAIEmbeddings, ChatOpenAI
    from langchain_community.vectorstores import FAISS
    from langchain_core.documents import Document
    from langchain.prompts import PromptTemplate
    from langchain.chains import ConversationalRetrievalChain
    from langchain.memory import ConversationBufferMemory
    from langchain.schema import AIMessage, HumanMessage, SystemMessage

    # Load .env
    load_dotenv()

    # Initialize LLM
    llm_2 = ChatOpenAI(model="gpt-4o")

    # Embeddings
    embedding_function = OpenAIEmbeddings(model="text-embedding-3-small")

    # Initialize FAISS vectorstore
    batch_size = 500
    vectorstore = None

    for i in range(0, len(documents), batch_size):
        batch = documents[i:i+batch_size]
        if vectorstore is None:
            vectorstore = FAISS.from_documents(batch, embedding_function)
        else:
            vectorstore.add_documents(batch)
        time.sleep(1)

    retriever = vectorstore.as_retriever(search_type="similarity", search_kwargs={"k": 3})

    # Convert conversation_messages to LangChain Message types
    formatted_messages = []
    for msg in conversation_messages:
        role = msg["role"]
        content = msg["content"]
```

```python
        content = msg["content"]
        if role == "user":
            formatted_messages.append(HumanMessage(content=content))
        elif role == "assistant":
            formatted_messages.append(AIMessage(content=content))
        elif role == "system":
            formatted_messages.append(SystemMessage(content=content))

    # Create memory and preload with conversation history
    memory = ConversationBufferMemory(
        memory_key="chat_history",
        return_messages=True
    )
    memory.chat_memory.messages = formatted_messages  # Inject messages

    # Build chain
    qa_chain = ConversationalRetrievalChain.from_llm(
        llm=llm_2,
        retriever=retriever,
        memory=memory,
        condense_question_prompt=custom_prompt_feed
    )

    # Invoke question
    query = prompt_question
    response = qa_chain.invoke({
    "question": query
   # "context": ""  # Or some string/summary if your prompt requires it
})
    print(response["answer"])
```

Figure 16: Second Function   using GPT 4.0 and four arguments and FAISS vector database

```python
# embeddings using Weaviate vector db with token-safe batching
def setup_credit_score_collection_edited(documents):
    import os
    from dotenv import load_dotenv
    from weaviate import WeaviateClient
    from weaviate.connect import ConnectionParams
    from weaviate.classes.config import Property, DataType, Configure
    from langchain_openai import OpenAIEmbeddings
    from langchain.schema import Document
    from langchain_weaviate import WeaviateVectorStore
    from tiktoken import get_encoding  # for GPT-4/3.5 tokenizer
    import warnings
    warnings.filterwarnings("ignore", category=DeprecationWarning)

    load_dotenv()

    # OpenAI tokenizer for text-embedding-3-small
    tokenizer = get_encoding("cl100k_base")

    def count_tokens(text):
        return len(tokenizer.encode(text))

    # Initialize Weaviate client
    client = WeaviateClient(
        connection_params=ConnectionParams.from_params(
            http_host="localhost", http_port=8080, http_secure=False,
            grpc_host="localhost", grpc_port=50051, grpc_secure=False
        ),
        skip_init_checks=True
    )

    client.connect()

    # Create collection if it doesn't exist
    if not client.collections.exists("CreditScore"):
        client.collections.create(
            name="CreditScore",
            properties=[
                Property(name="text", data_type=DataType.TEXT)
            ],
            vectorizer_config=Configure.Vectorizer.none()
        )
        print("✅ CreditScore collection created.")
```

```python
    )
    print("✅ CreditScore collection created.")
else:
    print("ℹ️ CreditScore collection already exists. Skipping setup.")

# Initialize embedding and vectorstore
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")

vectorstore = WeaviateVectorStore(
    client=client,
    index_name="CreditScore",
    text_key="text",
    embedding=embeddings
)

# Convert to LangChain documents
langchain_docs = [Document(page_content=doc.page_content, metadata=doc.metadata) for doc in documents]

# Token-safe batching logic
max_tokens = 250_000  # Leave room for headers or unexpected growth
batch = []
total_tokens = 0
uploaded = 0

for doc in langchain_docs:
    tokens = count_tokens(doc.page_content)

    if total_tokens + tokens > max_tokens:
        vectorstore.add_documents(batch)
        uploaded += len(batch)
        print(f"✅ Uploaded {len(batch)} documents in a batch.")
        batch = []
        total_tokens = 0

    batch.append(doc)
    total_tokens += tokens

if batch:
    vectorstore.add_documents(batch)
    uploaded += len(batch)
    print(f"✅ Uploaded final batch with {len(batch)} documents.")

print(f"📦 Total uploaded documents: {uploaded}")
client.close()
```

Figure 17:Third Function   using GPT 4.0 , one argument (document) and Weaviate Vector database

```python
[42]: #using gpt4.0 & weaviate db
def query_credit_score_bot_weaviate(custom_qa_prompt, prompt_question, conversation_messages):
    import os
    from dotenv import load_dotenv
    import weaviate
    from langchain_openai import ChatOpenAI, OpenAIEmbeddings
    from langchain_weaviate import WeaviateVectorStore
    from langchain.chains import ConversationalRetrievalChain
    import warnings
    warnings.filterwarnings("ignore", category=DeprecationWarning)

    def format_chat_history(messages):
        if isinstance(messages[0], tuple):
            return messages
        formatted = []
        last_user_msg = None
        for m in messages:
            if m["role"] == "user":
                last_user_msg = m["content"]
            elif m["role"] == "assistant" and last_user_msg:
                formatted.append((last_user_msg, m["content"]))
                last_user_msg = None
        return formatted

    load_dotenv()

    client = weaviate.connect_to_local(skip_init_checks=True)

    try:
        embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
        vectorstore = WeaviateVectorStore(
            client=client,
            index_name="CreditScore",
            text_key="text",
            embedding=embeddings
        )
        retriever = vectorstore.as_retriever()
        llm = ChatOpenAI(model="gpt-4o", temperature=0)
        formatted_history = format_chat_history(conversation_messages)
```

```
        llm = ChatOpenAI(model="gpt-4o", temperature=0)
        formatted_history = format_chat_history(conversation_messages)

        qa_chain = ConversationalRetrievalChain.from_llm(
            llm=llm,
            retriever=retriever,
            condense_question_prompt=custom_qa_prompt,
            return_source_documents=False
        )

        result = qa_chain({
            "question": prompt_question,
            "chat_history": formatted_history
        })

        return result

    finally:
        client.close()
```

Figure 18: Fourth Function using GPT 4.0 and three arguments

The fourth function which had GPT 4.0, with 3 arguments was used alongside the Weaviate vector database function and it responded to all prompts without missing key information from the embedded credit score dataset.

5.5. RAG based LLM Querying without prompt Optimization

All arguments for the above highlighted functions were applied in their default state.

The document argument was confirmed as shown below

Input

#View the created document

print(documents[0].page_content[:50])

Output

```
Customer ID: C02COQEVYU
CUST_ID: C02COQEVYU
INCOME
```

The prompt template argument was confirmed as shown below

from langchain.prompts import PromptTemplate

default_custom_prompt_style = PromptTemplate(

    input_variables=["question", "chat_history"],

    template="""

Given the following conversation and a follow up question, rephrase the follow up question to be a standalone question in two sentences at most.

Chat History:

{chat_history}

Follow Up Question: {question}

Standalone Question:

"""

)

The conversation messages argument was confirmed as shown below

default_conversation_messages = [

("Kindly respond to the Question", "")

]

5.6. Prompt Test 1

#prompts

prompt1 = """

How does this customer's (CZRA4MLB0P) credit score compare to the average credit score (586.7) and what debt category does the individual belong to?

"""

conversation_messages = [(prompt1, "")] # Empty assistant response for new conversation

RAG BASED LLM RESPONSE : PROMPT 1 NOT OPTIMIZED

response = query_credit_score_bot_weaviate (

custom_qa_prompt=default_custom_prompt_style,

prompt_question=prompt1,

conversation_messages=default_conversation_messages

)

print ("Response from bot:", response['answer'])

```
Response from bot: The credit score of customer CZRA4MLB0P is 589,
which is slightly above the average credit score of 586.7. This in
dividual belongs to the debt category 1.
```

5.7. Prompt Test 2 (Not Optimized)

#prompts

prompt2 = """

For Customer ID CZRA4MLB0P What is the total outstanding debt across all loans and credit lines and

How much of the individual's monthly income is allocated to debt repayment (Debt-to-Income Ratio - DTI)?

"""

RAG BASED LLM RESPONSE : PROMPT 2 NOT OPTIMIZED

response_2 = query_credit_score_bot_weaviate (

custom_qa_prompt=default_custom_prompt_style,

prompt_question=prompt2,

conversation_messages=default_conversation_messages

)

print ("Response from bot:", response_2['answer'])

```
Response from bot: The total outstanding debt for Customer ID CZRA4MLB0P is 680,837.
The Debt-to-Income Ratio (DTI) for this individual is 8.3637, which means that the d
ebt is approximately 836.37% of their monthly income.
```

5.8. Prompt Test 3 (Not Optimized)

#prompts

prompt3 = """

What are the travel expenses or inflow that affect this customer with Cust ID CZRA4MLB0P and How does this affect this customer's credit score?

"""

conversation_messages = [(prompt3, "")] # Empty assistant response for new conversation response_3 = query_credit_score_bot_weaviate (

custom_qa_prompt=default_custom_prompt_style,

prompt_question=prompt3,

conversation_messages=default_conversation_messages

)

print ("Response from bot:", response_3['answer'])

```
Response from bot: The travel expenses for the customer with Cust ID CZRA4MLB
0P are as follows:

- Total travel expenses over the last 12 months: $24,625
- Total travel expenses over the last 6 months: $3,859

The ratio of travel expenses to income (R_TRAVEL_INCOME) is 0.3025, to saving
s (R_TRAVEL_SAVINGS) is 0.2773, and to debt (R_TRAVEL_DEBT) is 0.0362.

Regarding the impact on the customer's credit score, the travel expenses them
selves do not directly affect the credit score. However, high travel expenses
relative to income, savings, and debt could indicate financial strain or poor
financial management, which might indirectly affect the credit score if it le
ads to increased debt or missed payments. In this case, the customer has a cr
edit score of 589, which is considered a poor credit score, and they have a h
igh debt-to-income ratio (R_DEBT_INCOME of 8.3637), which could be contributi
ng factors to the low credit score.
```

5.9. RAG based LLM Querying with prompt Optimization

All prompts initially passed were afterwards optimized with selected prompt techniques.

The code snippets and prompt customization of each argument is outlined below.

5.10. First Optimized Prompt

prompt1_Op = (

"Compare the credit score of Customer ID CZRA4MLB0P with the overall average credit score from the dataset. "

"Based on this, classify the customer debt category: Low, Medium, or High."

)

Prompt Optimization strategy here was based on the following:

Selected Prompt Type:

Emphasizes on clarity

Comparative Prompt + Risk Indicator Prompt

This combination works best because the goal is to compare an individual's credit score to an average (comparative).

Identifying the associated risk category (risk indicator).

The conversation message used

```
conversation_messages1_op = [
    ("""The dataset includes credit scores, from these credit score average credit score of 586.7 can be deduced. Also
    the dataset has a debt column that shows the debt owed by each customer, look into the debt column to deduce what category this customer falls""", "")
]
```

The Prompt Template used

```
from langchain.prompts import PromptTemplate


custom_prompt_style1_op = PromptTemplate(
    input_variables=["question", "chat_history"],
    template="""
You are a helpful credit data assistant. Using the available dataset, answer the user's question thoroughly.


The dataset includes fields like INCOME, DEBT, SAVINGS, CREDIT_SCORE, and other financial metrics.


Respond clearly using insights from similar records. If comparisons or averages are implied, include them.



Chat History:
{chat_history}


Follow Up Question: {question}


Standalone Question:
"""
)
```

The response from LLM

```
response_op = query_credit_score_bot_weaviate(
```

custom_qa_prompt=custom_prompt_style1_op,

prompt_question=prompt1_Op,

conversation_messages=conversation_messages1_op

)

Printing the response

print ("Response from bot:", response_op['answer'])

```
Response from bot: The credit score for Customer ID CZRA4MLB0P is 589, which is sl
ightly above the overall average credit score of 586.7. However, their debt amount
is 680,837, which is quite high. Given this information:

1. **Credit Score Comparison:**
   - The credit score of 589 is slightly above the average but not significantly h
igher.

2. **Debt Analysis:**
   - The debt amount of 680,837 is substantial.

Based on the credit score being slightly above average and the high debt amount, C
ustomer ID CZRA4MLB0P would likely fall into a "High" debt category.
```

5.11. Second Optimized Prompt

Second Optimized Prompt Using Multi-Style Prompting technique

prompt2_Op = (

"Step 1: Retrieve the total outstanding DEBT for Customer ID CZRA4MLB0P.\n"

"Step 2: Retrieve the monthly INCOME for the same customer.\n"

"Step 3: Calculate the Debt-to-Income Ratio (DTI = DEBT / INCOME).\n"

"Step 4: Provide the percentage of the individual’s monthly income allocated to debt repayment.\n"

"Step 5: Provide an interpretation of the DTI score.\n"

)

Note : Multi-Step Prompt LLM was instructed to follow an ordered reasoning path, which aligns with multi-step prompting.

The Conversation Messages used

conversation_messages2_op = [

(

"We're analysing customer financials. Use dataset fields like DEBT, INCOME, and SAVINGS to compute useful metrics like Debt-to-Income (DTI). Start by pulling these values for CZRA4MLB0P.",

"Understood. I'll extract DEBT and INCOME from the dataset, calculate DTI, and interpret the result."

)

]

Prompt_template_Used

```
from langchain.prompts import PromptTemplate


custom_prompt_style2_op = PromptTemplate(
    input_variables=["question", "chat_history"],
    template="""
You are a financial assistant skilled in analysing customer credit data.


Your task is to respond clearly using multi-step reasoning. The dataset includes DEBT, INCOME, CREDIT_SCORE, SAVINGS, and related fields. Use these to extract values, compute ratios, and provide clear conclusions.


Chat History:
{chat_history}


Follow Up Question: {question}


Answer in 3–5 sentences with a breakdown of your reasoning.
"""
)
```

Function Called

```
response2_op = query_credit_score_bot_weaviate (
    custom_qa_prompt=custom_prompt_style2_op,
    prompt_question=prompt2_Op,
    conversation_messages=conversation_messages2_op
```

)

Response Printed

print("Response from bot:", response2_op['answer'])

```
Response from bot: The information provided in your steps does not match t
he actual data for Customer ID CZRA4MLB0P. According to the dataset, the D
EBT for this customer is $680,837, and the annual INCOME is $81,404.

To calculate the Debt-to-Income Ratio (DTI) using the correct data:

1. **Step 1:** Retrieve the total outstanding DEBT for Customer ID CZRA4ML
B0P, which is $680,837.

2. **Step 2:** Retrieve the annual INCOME for the same customer, which is
$81,404. To find the monthly income, divide the annual income by 12: $81,4
04 / 12 = $6,783.67.

3. **Step 3:** Calculate the Debt-to-Income Ratio (DTI). The formula is DT
I = DEBT / (INCOME/12). Therefore, DTI = $680,837 / $6,783.67 ≈ 100.36.

4. **Step 4:** Convert the DTI into a percentage to understand the portion
of income allocated to debt repayment. DTI percentage = 100.36 * 100% = 10
,036%.

5. **Step 5:** Interpretation: A DTI of 10,036% indicates that the custome
r's total debt is over 100 times their monthly income, which is extremely
high. This suggests that the individual is significantly over-leveraged an
d may face severe difficulties in managing their debt obligations. Typical
ly, a DTI below 36% is considered manageable, so this customer may need to
explore debt reduction strategies or significantly increase their income t
o improve financial stability.
```

5.12. Third Optimized Prompt

Third Optimized Prompt implemented using the Chain of thought prompt

prompt3_op = """

Analyse the travel-related expenses and income for the customer with Cust ID CZRA4MLB0P.

First, identify if the customer has any notable travel-related DEBT, SAVINGS, or INCOME records.

Then, step-by-step, reason how each factor (debt, savings, income) could potentially affect the customer's credit score.

Finally, conclude with a brief evaluation of the overall credit impact for this customer.

"""

Selected Prompt Type:

Chain of thought prompt

Chain-of-thought prompting requires language models to present reasoning steps, or thoughts, before giving a final answer. This technique is valuable for complex reasoning tasks and helps reduce errors by processing the reasoning step by step.

Conversation message used

conversation_messages3_op = [

{"role": "system", "content": """The dataset contains fields such as CREDIT_SCORE, INCOME, DEBT, SAVINGS, and others.

The average CREDIT_SCORE across the dataset is approximately 586.7.

Important context:

- DEBT column shows total outstanding debts (including travel-related debts).

- INCOME may include travel income if relevant.

- SAVINGS indicates financial discipline, including travel savings.

Use this context to first find where the customer falls in debt or income categories compared to others,

and reason about the intermediate factors before concluding the effect on credit score. Give a concise response"""}

]

Prompt Template Used

from langchain.prompts import PromptTemplate

custom_prompt_style3_op = PromptTemplate(
input_variables=["question", "chat_history"],
template="""
You are a financial credit analyst assistant specialized in customer behaviour.

Your task is to assist the user in analysing financial datasets to deduce credit score impacts.

Ensure your responses are concise without unnecessary information

The dataset includes columns like INCOME, DEBT, SAVINGS, CREDIT_SCORE, and others.

Important Instructions:

1. Break down your reasoning step-by-step (Chain-of-Thought).
2. First, identify the relevant data fields.
3. Compare or benchmark where necessary (e.g., against averages).
4. Explain intermediate deductions logically.
5. Finally, summarize the credit score implications clearly.

Use examples from similar customers if it helps.

Chat History:
{chat_history}

Follow-Up Question:
{question}

Standalone Version of the Question:
"""
)

Function Called

```
response3_op = query_credit_score_bot_weaviate (
    custom_qa_prompt=custom_prompt_style3_op,
    prompt_question=prompt3_op,
    conversation_messages=conversation_messages3_op
)
```

Response Printed

```
print("Response from bot:", response3_op['answer'])
```

```
Response from bot: For the customer with Cust ID CZRA4MLB0P, let's analyse the tra
vel-related expenses and their relationship with debt, savings, and income:

1. **Travel-Related Expenses:**
   - Total travel expenses over the last 12 months: $24,625
   - Total travel expenses over the last 6 months: $3,859
   - Ratio of travel expenses to income (R_TRAVEL_INCOME): 0.3025
   - Ratio of travel expenses to savings (R_TRAVEL_SAVINGS): 0.2773
   - Ratio of travel expenses to debt (R_TRAVEL_DEBT): 0.0362

2. **Notable Records:**
   - **Income:** $81,404
   - **Savings:** $88,805
   - **Debt:** $680,837

3. **Analysis of Factors:**

   - **Debt:**
     - The customer's debt is significantly high at $680,837, with a debt-to-incom
e ratio (R_DEBT_INCOME) of 8.3637. This indicates that the customer has a substant
ial amount of debt compared to their income.
     - The travel expenses relative to debt (R_TRAVEL_DEBT) are low at 0.0362, sug
gesting that travel expenses are a small portion of the overall debt. However, the
high overall debt level can negatively impact the credit score.

   - **Savings:**
     - The customer has savings of $88,805, which is slightly higher than their in
come, with a savings-to-income ratio (R_SAVINGS_INCOME) of 1.0909.
     - The travel expenses relative to savings (R_TRAVEL_SAVINGS) are 0.2773, indi
cating that a moderate portion of savings is used for travel. While this shows som
e financial flexibility, the high debt level overshadows the positive impact of sa
vings.

   - **Income:**
     - The income is $81,404, and the travel expenses relative to income (R_TRAVEL
_INCOME) are 0.3025, meaning that a significant portion of income is spent on trav
el.
     - While the income level is reasonable, the high debt-to-income ratio suggest
s that the customer may struggle to manage their debt effectively, which can adver
sely affect the credit score.

4. **Overall Credit Impact:**
   - The customer's credit score is 589, which is considered a poor credit score.
The high level of debt relative to income and savings is a significant factor cont
ributing to this low score.
   - Despite having some savings, the overwhelming debt burden and the substantial
portion of income spent on travel expenses indicate financial stress, which negati
vely impacts the credit score.
   - The customer has defaulted (DEFAULT: 1), further indicating financial difficu
lties and contributing to the poor credit score.

In conclusion, while the customer has some savings and a reasonable income, the hi
gh debt level and significant travel expenses relative to income contribute to a p
oor credit score. The overall financial situation suggests a need for better debt
```

5.13. Evaluation of Output

Evaluation of output was executed using both qualitative and quantitative approach. A comparison of the RAG based LLM outputs for both standard/ordinary prompts and optimized prompts were analysed using the following evaluation metrics:

Qualitative Evaluation using ChatGPT

- Relevance to Prompt
- Data Accuracy
- Reasoning Quality
- Faithfulness to source Documents
- Clarity & Structure

EVALUATION & COMPARISON OF DIRECT PROMPT AND OPTIMISED PROMPTS FOR THE FIRST PROMPT

Criteria	Response(Response 1)	Response_op(Response 2)	Comparison Notes
Relevance to Prompt	Partially relevant. Only gives credit score and debt category.	Highly relevant. Discusses credit score *and* debt amount explicitly.	**Response 2** gives a fuller, more targeted answer.
Data Accuracy	Correct credit score and slight above-average comment. Debt category mentioned but no debt amount shown.	Correct credit score, average comparison, and actual debt amount ($680,837) provided.	**Response 2** shows better use of available data.
Reasoning Quality	Very basic — no real analysis, just states results.	Step-by-step reasoning: credit score compared, debt analyzed, inference about category made.	**Response 2** demonstrates *actual reasoning* and thought process.
Faithfulness to Source Documents	Faithful but shallow — no fabricated information, but little use of source richness.	Faithful and richer — uses both the credit score and debt fields to reason.	Both are faithful, but **Response 2** extracts more value.
Clarity	Clear, but extremely brief — almost too little.	Very clear, well-organized with points and brief explanation under each.	**Response 2** is clearer thanks to structure and labeling.
Structure	Minimal — just one or two sentences.	Organized into numbered sections: Credit Score Comparison, Debt Analysis.	**Response 2** is **much better structured** for understanding.

Figure 19: Qualitive evaluation of Unoptimized and optimized prompt 1 using ChatGPT

Quantitative Evaluation

Measuring Precision using Bilingual Evaluation Understudy

```
#Evaluation using Blue on Standard or Ordinary Prompt 1
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

reference = ["""Compared to the average credit score of 586.7, CZRA4MLB0P's credit
                score is slightly above average.The dataset categorizes this debt as \"high\" """]
response = """The credit score of customer CZRA4MLB0P is 589,
                which is slightly above the average credit score of 586.7. This individual belongs to the debt category 1."""

score = sentence_bleu(
    [ref.split() for ref in reference],
    response.split(),
    smoothing_function=SmoothingFunction().method4
)
print(f"BLEU Score: {score:.4f}")

BLEU Score: 0.2151
```

```
#Evaluation using Blue on Optimized Prompt 1
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

reference = ["""Compared to the average credit score of 586.7, CZRA4MLB0P's
                credit score is slightly above average.The dataset categorizes this debt as \"high\" """]
response_op = """The credit score for customer CZRA4MLB0P is 589.- Compared to the average credit score of 586.7,
                    CZRA4MLB0P's credit score is slightly above average.
                    The debt amount for customer CZRA4MLB0P is 680,837.
   - The dataset categorizes this debt as "high" since the `CAT_DEBT` value is 1, indicating a significant level of debt.
."""

score = sentence_bleu(
    [ref.split() for ref in reference],
    response_op.split(),
    smoothing_function=SmoothingFunction().method4
)
print(f"BLEU Score: {score:.4f}")

BLEU Score: 0.3338
```

Figure 20: Measurement of Precision for RAG based LLM response Prompt 1 using BLEU (Optimized prompts with BLEU score of 0.3338 & unoptimized prompts with BLEU score of 0.2115

Evaluation Criteria	Response 1	Response 2	
Relevance to Prompt	High-Directly answers the prompt with key values	High -Fully addresses the prompt.	Response 2 provides better alignment with prompt and detailed coverage
Data Accuracy	High – Accurately states DEBT and DTI.	High – Accurately presents DEBT and INCOME with DTI.	Response 2 delivers more context and interpretive accuracy
Reasoning Quality	Medium – Mentions interpretation but lacks detail.	High – Detailed reasoning and contextual interpretation.	Response 2 offers more comprehensive
Faithfulness to Source Documents	Medium – Assumes values without step-by-step validation.	High – Clearly references specific values from dataset.	Response 2 explains data origin and interpretation more effectively
Clarity & Structure	Medium – Concise but lacks breakdown.	High – Well-structured with explanation and breakdown.	Response 2 excels in readability and format

Figure 21: Qualitive evaluation of Unoptimized and optimized prompt 2 using ChatGPT

```
#Evaluation using Blue on Standard or Ordinary Prompt 2
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

reference = ["""The total outstanding debt is $680,837 ,the income of the individual's total income is $81,404.
                The (Debt-to-Income Ratio - DTI) of the individual is 8.36 and 836% of the individual's income is used to service
                debts and this is considered risky """]
response = """The total outstanding debt is 680,837 , The Debt-to-Income Ratio (DTI) for this individual is 8.3637,
                which means that the debt is approximately 836.37% of the individual's monthly income."""

score = sentence_bleu(
    [ref.split() for ref in reference],
    response.split(),
    smoothing_function=SmoothingFunction().method4
)
print(f"BLEU Score: {score:.4f}")
```

```
BLEU Score: 0.1295
```

```
#Evaluation using Blue on Optimized Prompt 2
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

reference = ["""The total outstanding debt is $680,837 ,the income of the individual's total $81,404.
                The (Debt-to-Income Ratio - DTI) of the individual is 8.36 and 836% of the individual's income is used to service
                debts and this is considered risky """]
response_op = """The Retrieve DEBT is $680,837 ,the retrieved INCOME of the individual is $81,404
                The (Debt-to-Income Ratio - DTI) of the individual is 8.36 and
                A DTI of 836% indicates an extremely high level of debt relative to income """

score = sentence_bleu(
    [ref.split() for ref in reference],
    response_op.split(),
    smoothing_function=SmoothingFunction().method4
)
print(f"BLEU Score: {score:.4f}")
```

```
BLEU Score: 0.3320
```

Figure 22: Measurement of Precision for RAG based LLM response Prompt 2 using BLEU (Optimized prompts with BLEU score of 0.3320 & unoptimized prompts with BLEU score of 0.1295)

Criteria	Response3	Response3_op	Comments
Relevance to Prompt	Good	Excellent	Both responses answer the prompt, but Response 2 follows the multi-step "analyse → reason → conclude" format better.
Data Accuracy	Good	Excellent	Both responses report correct figures, but Response 2 includes additional important financial metrics (Income, Debt, Savings) not mentioned in Response 1.
Reasoning Quality	Moderate	High	Response 1 offers basic analysis ("could affect credit score"), but Response 2 performs deep reasoning (evaluating relationships between debt, savings, and travel expenses).
Faithfulness to Source Documents	Good	Excellent	Both responses are faithful to the dataset, but Response 2 references the `DEFAULT` field and ratio calculations more thoroughly.
Clarity	Good	Very Good	Response 1 is simpler but a bit abrupt. Response 2 is clear, well-explained, and properly breaks down ideas.
Structure	Basic	Excellent	Response 1 is a paragraph. Response 2 has a clean logical structure: headings, numbered sections, bullet points.

Figure 23: Qualitive evaluation of Unoptimized and optimized prompt 3 using ChatGPT

```python
#Evaluation using Blue on Standard or Ordinary Prompt 3
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

reference = ["""The travel expenses are
                Total travel expenses over the last 12 months: $24,625
                 Total travel expenses over the last 6 months: $3,859
                The customer's poor credit score of 589 is primarily due
                to high debt relative to their income and savings. Although travel-related expenses
                contribute to overall spending, they are not the main factor. Ultimately, the high debt
                burden outweighs any positive effects from savings or income, making it the key driver of the low credit score.
 """]
response = """he customer with Cust ID CZRA4MLB0P has travel expenses of 24,625 over the past 12 months and 3,859 over the past 6 months.
The ratio of travel expenses to income (R_TRAVEL_INCOME) is 0.3025, to savings (R_TRAVEL_SAVINGS) is 0.2773, and to debt (R_TRAVEL_DEBT) is 0.0362.
These travel expenses, along with other financial factors, can impact the customer's credit score.
The customer's credit score is 589, which is considered a poor credit score. High travel expenses
relative to income and savings can contribute to financial strain, especially when combined with
a high debt-to-income ratio (R_DEBT_INCOME of 8.3637) and a high debt-to-savings ratio (R_DEBT_SAVINGS of 7.6667).
These factors suggest that the customer may be over-leveraged, which can negatively affect their credit score. Additionally,
the customer has a default status of 1,
indicating a history of missed payments or defaults, further impacting their credit score."""

score = sentence_bleu(
    [ref.split() for ref in reference],
    response.split(),
    smoothing_function=SmoothingFunction().method4
)
print(f"BLEU Score: {score:.4f}")
```

```
BLEU Score: 0.0197
```

```python
#Evaluation using Blue on Optimized Prompt 3
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

reference = ["""The travel expenses are
            Total travel expenses over the last 12 months: $24,625
             Total travel expenses over the last 6 months: $3,859
                The customer's poor credit score of 589 is primarily due
                to high debt relative to their income and savings. Although travel-related expenses
                contribute to overall spending, they are not the main factor. Ultimately, the high debt
                burden outweighs any positive effects from savings or income, making it the key driver of the low credit score. """]

response_op = """Travel-Related Expenses:**
   - Total travel expenses over the last 12 months: $24,625
   - Total travel expenses over the last 6 months: $3,859
   - Ratio of travel expenses to income (R_TRAVEL_INCOME): 0.3025
   - Ratio of travel expenses to savings (R_TRAVEL_SAVINGS): 0.2773
   - Ratio of travel expenses to debt (R_TRAVEL_DEBT): 0.0362

2. **Notable Financial Records:**
   - **Income:** $81,404
   - **Savings:** $88,805
   - **Debt:** $680,837
The customer's credit score is 589, which is considered a poor credit score. This is likely due to the high levels of debt
relative to income and savings.
   - The travel-related expenses, while not excessively high, contribute to the overall expenditure but are not the primary
   factor affecting the credit score.
   - The combination of high debt and moderate income, despite having some savings, results in a negative impact on the credit score.

In conclusion, the customer's high debt levels are the most significant factor negatively affecting their credit score,
outweighing the potential positive impact of their savings and income.
The travel-related expenses are a part of the overall expenditure but do not significantly alter the credit impact."""

score = sentence_bleu(
    [ref.split() for ref in reference],
    response_op.split(),
    smoothing_function=SmoothingFunction().method4
)
print(f"BLEU Score: {score:.4f}")
```

```
BLEU Score: 0.1421
```

Figure 24: Measurement of Precision for RAG based LLM response Prompt 3 using BLEU (Optimized prompts with BLEU score of 0.1421 & unoptimized prompts with BLEU score of 0.0197)

Chapter 6

6. Conclusion

6.1. Ethical Considerations and Risk Management in LLM-Based Financial Systems

Large Language Models (LLMs) can inadvertently reflect or even amplify biases embedded within the data on which they were trained, making this a critical concern in the financial sector. Such algorithmic bias has the potential to produce unfair outcomes in core processes like credit approvals, risk evaluations, and customer interactions. These outcomes can expose institutions to regulatory penalties, legal challenges, and erosion of customer confidence(Fan, M., 2024).

The application of Large Language Models (LLMs) in financial decision-making introduces several ethical considerations, particularly in regulated domains such as credit scoring. Bias in training data can inadvertently influence model outputs, potentially leading to unfair or discriminatory decisions that affect individuals' financial opportunities. For instance, attributes like employment type or geographic location which is common in credit datasets could become proxies for protected characteristics if not carefully managed during Exploratory Data Analysis (EDA). Therefore, it is crucial to ensure fairness from the earliest stages of the workflow (OpenAI,2025). To manage bias, risk and promote ethical considerations for this project, during EDA of the Kaggle-sourced credit score dataset, careful attention was paid to identifying and mitigating such risks by reviewing variable distributions, checking for outliers, and assessing data completeness to minimize biased learning from skewed or incomplete inputs.

Transparency and explainability of LLM outputs are equally critical, especially when outputs inform high-stakes financial decisions. The Retrieval-Augmented Generation (RAG) framework, while powerful, introduces complexity that may obscure how certain responses are generated. To address this, the RAG pipeline incorporated modular, interpretable components—such as vector database selection and prompt engineering strategies making the process more traceable and auditable. Prompt optimization techniques like Chain-of-Thought and Comparative Prompting were not only used to enhance response accuracy but also to improve reasoning transparency, allowing users to understand the logic behind generated insights. Additionally, storing retrieved documents in well-organized vector stores like FAISS and Weaviate supported traceability, making it possible to attribute specific outputs to corresponding input segments.

Another ethical concern is the risk of hallucination, where LLMs generate plausible but factually incorrect information. This was mitigated by embedding a verification loop within the experimental methodology: comparing outputs from ordinary prompts with those derived from optimized prompts and benchmarking them against known ground truths. Evaluation using both qualitative analysis and BLEU-based quantitative metrics further ensured that the generated insights were factually grounded and contextually accurate. By combining prompt engineering, transparency-focused workflows, and evaluation metrics, this study emphasizes not only the technical efficacy of LLMs in credit analysis but also a responsible and ethical approach to their deployment in real-world financial systems.

6.2 Summary of Project Contributions & Critical Analysis

Several steps were taken to achieve the objectives of this project. Firstly, a systematic Retrieval-Augmented Generation (RAG) pipeline was designed and implemented using a structured row-wise document retrieval technique tailored to the nature of the data and the problem domain. The storage process was finalized using the Weaviate vector database, as it demonstrated superior search algorithms and embedding capabilities. This database can be extended to future projects requiring effective document retrieval. Two Large Language Models (LLMs) were evaluated for their performance and suitability. Based on response accuracy and contextual understanding, GPT-4.0 emerged as the more effective model and was selected for further experimentation.

The integration of RAG components, LLM, and prompt designs was successfully achieved by developing a pipeline function that allowed for the injection of three distinct arguments to support robust customization and testing of different prompt styles. To facilitate seamless interaction among components, three modular pipeline functions were developed to connect prompt formulation, embedding retrieval, and LLM query execution. These functions were benchmarked, and the most effective configuration was identified as the combination of GPT-4.0, the Weaviate vector database, and LangChain's ConversationalRetrievalChain. This configuration yielded high-quality responses, efficient execution times, and overall system reliability.

To enable optimal prompt customization, three distinct prompt types were created: a Custom Prompt Template, a Standard Prompt, and a Conversation Prompt. From a pool of candidate queries, three baseline prompts were initially submitted to the LLM to observe its default behaviour. These prompts were then re-engineered using advanced prompt engineering techniques and evaluated for improvements in clarity, relevance, and factual grounding. The optimization strategies included Comparative Prompting with Risk Indicators, Multi-Style Prompting, and Chain-of-Thought Prompting. These methods significantly enhanced the interpretability and reliability of the credit score insights generated by the RAG-based LLM system.

Finally, outputs from both ordinary and optimized prompts were evaluated using a combination of qualitative and quantitative methods. Qualitatively, while ordinary prompts delivered reasonable accuracy due to support from embedded datasets, optimized prompts consistently outperformed them across all qualitative indicators. Metrics such as relevance, data accuracy, reasoning quality, faithfulness to source, and clarity all showed higher ratings for optimized prompts in all three tests. These observations were further substantiated through quantitative evaluation using BLEU scores, where optimized prompts achieved an average BLEU score of 0.27, with particularly insightful outputs generated using multi-step and chain-of-thought prompt techniques. All two of the research questions have been partly answered with chain of thoughts & multi-steps prompts has shown great interpretability and response optimization while RAG implementation has proven to support the accuracy LLMs even for ordinary unoptimized prompts.

Both research questions have been partially addressed. The use of Chain-of-Thought and Multi-Step prompting techniques demonstrated significant improvements in interpretability, traceability, transparency, and improved response quality. Additionally, the implementation of the Retrieval-Augmented Generation (RAG) framework effectively enhanced the accuracy of the LLM outputs, even when using ordinary, unoptimized prompts.

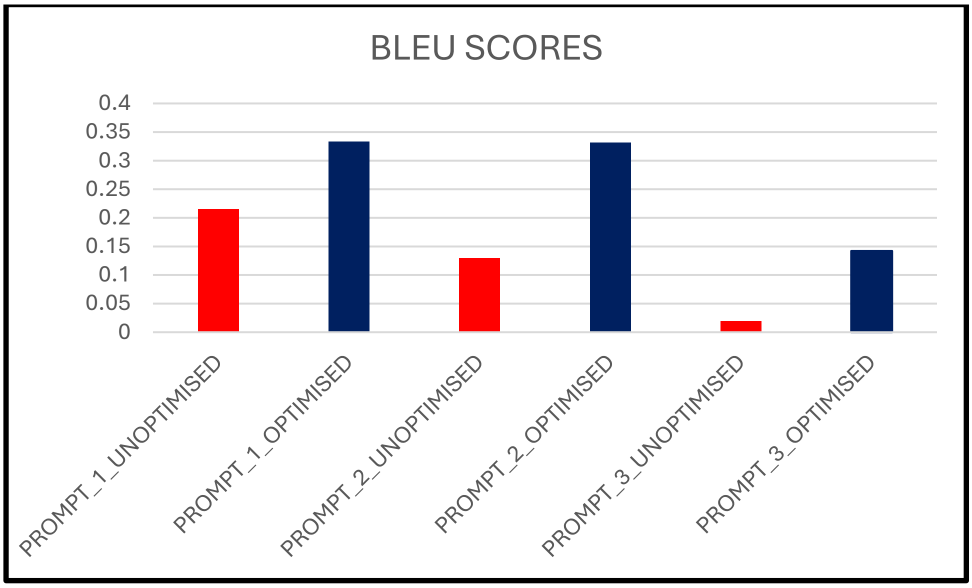

Figure 25: BLEU Scores for Unoptimized and Optimized prompts pass to the RAG based LLM

6.3. Limitations & Future Work

While this study presents a structured approach to using optimized prompt engineering in RAG-based LLM systems for credit score insights, several limitations highlight important areas for future improvement. First, the Kaggle credit score dataset, though useful for experimentation, lacks the depth, diversity, and real-world complexity of proprietary financial datasets. It does not include longitudinal or transactional data that would enable more accurate modeling of credit behaviour over time. Future work should explore the use of enriched datasets with behavioural and temporal dimensions to improve contextual understanding.

A second limitation concerns the evaluation methodology. While BLEU scores were employed for quantitative assessment and supplemented by qualitative criteria such as clarity, relevance, and faithfulness, these metrics could be further enhanced with advanced alternatives such as BERTScore or FactScore to better evaluate the reasoning capabilities and semantic accuracy of LLMs in financial analysis. Additionally, financial subject matter experts could be engaged to provide domain-specific reviews of LLM outputs.

Finally, while both FAISS and Weaviate vector databases were evaluated under basic conditions, their scalability and performance under enterprise-grade loads remain unexplored. Future work could involve stress-evaluating these vector databases under concurrent queries and dynamic data updates.

In conclusion, this project provides a strong foundation for optimizing prompts used in RAG-based LLMs but underscores the need for deeper exploration into fairness, explainability, data robustness, and scalable deployment to ensure responsible and trustworthy application in financial contexts.

References

Arora, S., Narayan, A., Chen, M.F., Orr, L., Guha, N., Bhatia, K., Chami, I. and Re, C. (2022) *Ask me anything: A simple strategy for prompting language models.*

Arslan, M., Ghanem, H., Munawar, S. and Cruz, C. (2024) 'A Survey on RAG with LLMs', *Procedia Computer Science,* 246, pp. 3781–3790.

Babaei, G. and Giudici, P. (2024) 'GPT classifications, with application to credit lending', *Machine Learning with Applications,* , pp. 100534.

Cachay, S.R., Boecking, B. and Dubrawski, A. (2021) 'Dependency structure misspecification in multi-source weak supervision models', *arXiv preprint arXiv:2106.10302,* .

Cao, L., Yang, Q. and Yu, P.S. (2021) 'Data science and AI in FinTech: An overview', *International Journal of Data Science and Analytics,* 12(2), pp. 81–99.

Dastile, X., Celik, T. and Potsane, M. (2020) 'Statistical and machine learning models in credit scoring: A systematic literature survey', *Applied Soft Computing,* 91, pp. 106263.

Fan, M. (2024) *LLMs in Banking: Applications, Challenges, and Approaches.* pp. 314.

Fan, W., Zhao, Z., Li, J., Liu, Y., Mei, X., Wang, Y., Tang, J. and Li, Q. (2023) 'Recommender systems in the era of large language models (llms)', *arXiv preprint arXiv:2307.02046,* .

Feng, D., Dai, Y., Huang, J., Zhang, Y., Xie, Q., Han, W., Lopez-Lira, A. and Wang, H. (2023) 'Empowering many, biasing a few: Generalist credit scoring through large language models', *arXiv preprint arXiv:2310.00566,* .

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H. and Wang, H. (2023) 'Retrieval-augmented generation for large language models: A survey', *arXiv preprint arXiv:2312.10997,* 2, pp. 1.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W. and Rocktäschel, T. (2020) 'Retrieval-augmented generation for knowledge-intensive nlp tasks', *Advances in neural information processing systems,* 33, pp. 9459–9474.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J. and Schmidt, D.C. (2023) 'A prompt pattern catalog to enhance prompt engineering with chatgpt', *arXiv preprint arXiv:2302.11382,* .

Xu, L., Zhang, J., Li, B., Wang, J., Cai, M., Zhao, W.X. and Wen, J. (2024) 'Prompting Large Language Models for Recommender Systems: A Comprehensive Framework and Empirical Analysis', *arXiv preprint arXiv:2401.04997,* .