Bachelor's thesis

AUTOMATICALLY DRAFTING AN ENGAGING BLOG POST BASED ON AN ACADEMIC PUBLICATION

Kanstantsin Downar

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: doc. Mgr. Viliam Lisý, MSc., Ph.D.
May 10, 2025

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Automatically drafting an engaging blog post based on an academic publication
Student:	Kanstantsin Downar
Supervisor:	doc. Mgr. Viliam Lisý, MSc., Ph.D.
Study program:	Informatics
Branch / specialization:	Artificial Intelligence 2021
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2025/2026

Instructions

Generative AI agents are expected to gradually automate routine computer tasks. One such task is producing engaging content oriented to the general public from technical scientific publications.

The student will:
- Review existing recommendations for writing engaging blog posts based on scientific papers
- Review existing works on predicting virality/engagement of a social network post
- Review the available Large Language Models suitable for creating LLM agents
- Create a dataset of papers, corresponding blog posts and the estimates of their success/engagement
- Create a prediction model for predicting the engagement of a blog post based on an LLM
- Design and implement a system that will use LLMs to automatically turn a given scientific paper into a blog post that follows the recommendations and optimizes the engagement
- Evaluate and discuss the strengths and weaknesses of the implemented system

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 15 January 2025 in Prague.

Citation of this thesis: Downar Kanstantsin. *Automatically drafting an engaging blog post based on an academic publication*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2025.

I would like to express my sincere gratitude to my thesis supervisor, doc. Mgr. Viliam Lisý, MSc., Ph.D., for his valuable guidance, insightful feedback, and support throughout the course of this work.
I also wish to thank my family and friends for their constant support and understanding during the entire period of writing this thesis. Their encouragement meant a great deal to me.

Declaration

Abstract

This study explores the potential applications of large language models (LLM) for processing scientific publications and automatically generating engaging blog posts which are written based on the original research articles. The primary objective of the research is to develop a system that utilizes LLMs as a key computational element for efficiently transforming scientific texts into an accessible format. To enhance generation quality, this work tests prompt engineering, with the potential use of additional modules to extend the query context, as the main method for optimizing interaction with the language model. As a result, a fully functional system has been developed that takes a scientific article as input and generates an informative and engaging blog post based on it. This approach contributes to the popularization of science and facilitates the dissemination of complex scientific ideas to a broader audience.

Keywords LLM-based agent, text generation, large language models, LLM, prompt, prompt engineering, system

Abstrakt

Tato práce se zabývá zkoumáním možností využití velkých jazykových modelů (LLM) pro zpracování vědeckých publikací a automatickou generaci poutavých blogových příspěvků, které jsou napsány na základě původních vědeckých článků. Hlavním cílem výzkumu je vývoj systému, který využívá LLM jako klíčový výpočetní prvek pro efektivní převod vědeckého textu do srozumitelného formátu. Pro zlepšení kvality generování tato práce testuje prompt engineering s možným využitím dalších modulů pro rozšíření kontextu dotazu jako hlavní metodu pro optimalizaci interakce s jazykovým modelem. Výsledkem práce je plně funkční systém, který přijímá vědecký článek jako vstup a na jeho základě generuje informativní a poutavý blogový příspěvek. Tento přístup přispívá k popularizaci vědy a usnadňuje šíření složitých vědeckých myšlenek širšímu publiku.

Klíčová slova agent založený na LLM, generování textu, velké jazykové modely, LLM, prompt, prompt engineering, systém

Contents

List of Figures

List of Tables

List of Algorithms

List of abbreviations

AI	Artificial intelligence
API	Application programming interface
CoT	Chain-of-Thought
FAISS	Facebook AI Similarity Search
GKP	Generated Knowledge Prompting
LLM	Large Language Model
LTM	Long-term memory
RAG	Retrieval-Augmented Generation
RPD	Requests per day
RPM	Requests per minute
TPM	Tokens per minute

Chapter 1
Introduction

1.1 Motivation

Scientific articles are one of the most important sources of information, providing insight into new research and discoveries in modern science. However, scientific articles are typically structured in a complex manner, written in a formal academic style, and filled with specialized terminology. As a result, such publications remain inaccessible to a broad audience. This group includes not only general readers, but also researchers from related disciplines, who could also benefit from the information presented.

Due to these challenges, a gap arises between the target audience of a publication and a wider readership, which prevents the popularization and dissemination of scientific knowledge.

Thus, developing a system for the automatic generation of high-quality and readable blog posts based on scientific articles can significantly improve access to scientific information, stimulate public interest in science, and create an intermediary bridge between experts and readers who may not have deep knowledge of the field.

1.2 Goals

The main objectives of this study are:

- to explore the potential of LLMs in generating blog posts based on scientific articles,
- to test various prompt engineering techniques for LLMs to maximize model performance,
- to develop a system that takes a scientific article as input, generates a highly engaging blog post using internal mechanisms, and returns the generated blog to the user,

- to conduct experiments on the implemented system using a publicly available data.

1.3 Structure of the thesis

This work is structured as follows.

Chapter 2 reviews existing recommendations for writing engaging blog posts based on scientific papers.

Chapter 3 presents the theoretical aspects of all elements used in this study.

Chapter 4 describes the implementation methods of the system introduced in the previous section, along with the datasets used.

Chapter 5 presents the conducted experiments and their results.

Chapter 6 serves as the conclusion, summarizing the overall findings of the study.

Chapter 2

Recommendations for Engaging Blogs

Creating an engaging blog based on a scientific article requires a careful balance between maintaining scientific accuracy and ensuring accessibility for a wider audience. Based on Sussex [1], admin [2], Springer Nature [3], and Communication [4], the following key recommendations can help transform an academic study into an engaging blog post:

Strong introduction: a successful blog begins with a compelling introduction. It is essential to start with a statement that captures attention, such as a thought-provoking question, an interesting fact, or a real-life example related to the research. The title should also be clear, concise, and intriguing, encouraging readers to continue exploring the content.

Simplicity and clarity: scientific papers often contain dense information and complex data. When translating these into a blog, it is important to simplify the content. Sentences should be short and direct to ensure clarity and ease of understanding.

Logical structure: the blog should be organized in a manner that enhances readability. Breaking the content into smaller, manageable sections with the help of subheadings helps guide the reader through the research. Each section should address a specific aspect of the study, ensuring a coherent flow. Using lists, bullet points, and brief paragraphs improves the accessibility and comprehension of the text.

Emphasis on key takeaways: the primary findings of the research should be clearly communicated to the readers. The focus should be on the essential points, avoiding excessive details. The takeaways should be relevant and applicable to real-world situations.

Reader engagement: encouraging interaction can enhance the blog's effectiveness. This can be achieved by asking open questions or offering the opportunity for readers to share their thoughts through comments. Including a call to action, such as encouraging readers to explore the full research paper or share the blog, can increase engagement. Connecting the research to real-world applications or its potential impact on people's lives helps make the blog more relatable. Providing practical insights or advice can further enhance its value. Conclusion with a question or suggestion for action can encourage continued discussion and participation.

The optimal length: the length of the blog is a crucial factor to consider. For regular posts, it is recommended to aim for a word count between 750 and 1,000 words, ensuring that the content is detailed enough while remaining engaging.

Chapter 3

Theoretical background

3.1 Large Language Models

3.1.1 Transformers

According to Vaswani et al. [5], the transformer is a neural network architecture based solely on the attention mechanism (self-attention), without using recurrent or convolutional layers.

In a high-level overview, the operation of a transformer can be described as an iterative next-token prediction, considering the entire context provided at each step. Tokens are generated until a termination condition is met (such as generating an end-of-sequence token, reaching a maximum output length, or other constraints). A more detailed description of the architecture is provided in the next section.

Building on this architecture, large language models (LLMs) are typically transformers with hundreds of billions of parameters, trained on massive datasets [6]. This is what gives them their name.

3.1.2 Transformer architecture

This section focuses on the decoder-only transformer architecture, which serves as the foundation for most LLMs, including models used in this thesis.

The schema in figure 3.1 presents the transformer architecture described by Vaswani et al. [5]. The block shown on the right side represents the **decoder stack** with N layers. Each decoder layer consists of several key components described in more detail below.

Multi-Head Attention: multi-head attention is an extension of the scaled dot-product attention mechanism that allows the model to analyze different aspects of input data in parallel. Instead of a single attention mechanism,

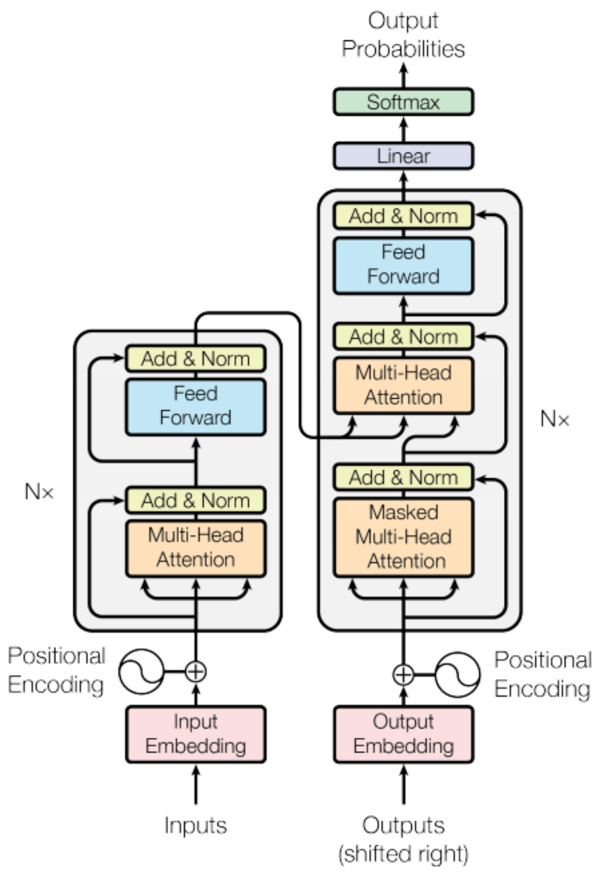


Figure 3.1 Transformer architecture [5]

h parallel attention heads are used, each learning different dependencies between tokens. The basic attention mechanism is computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{3.1}$$

The matrices Q (queries), K (keys), and V (values) contain parameters that change during the training of the model. [5]

Position-wise Feed-Forward Networks: this component consists of a fully connected feed-forward network that is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between. [5]

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2. \tag{3.2}$$

Add & Norm: this combination of two operation applies after each sub-layer performed. First, applies a residual connection, where the input to a sub-layer (such as multi-head self-attention or feed-forward network) is added to its output. After this, layer normalization is performed to stabilize the training process. [5]

3.2 Prompt Engineering

Prompt engineering refers to the systematic design and optimization of input prompts to guide the responses of LLMs, ensuring accuracy, relevance, and coherence in the generated output [7]. A prompt, in turn, is a textual instruction that facilitates user interaction with LLM, allowing the user to formulate a query to which the model generates a response based on its pre-trained knowledge and information processing algorithms.

This section will describe the fundamental techniques of prompt engineering. It will cover both basic methods and more sophisticated and advanced strategies that enhance the precision, coherence, and adaptability of language model responses. Additionally, practical applications and examples will be provided to illustrate their effectiveness in real-world scenarios.

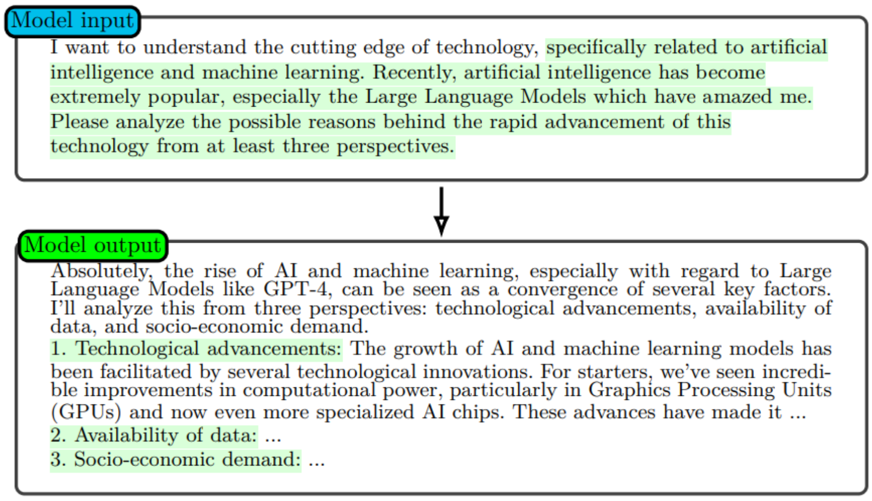
3.2.1 Basic techniques

3.2.1.1 Formulating Instructions

Instruction prompting plays a key role in guiding text generation by language models. Simple commands such as “Write”, “Classify”, “Summarize”, “Translate” effectively direct the model to perform basic tasks [8]. However, if an instruction is too general, the model faces a wide range of possible interpretations, leading to overly broad or ambiguous results. This occurs because, without clear context and additional clarifications, an LLM interprets the task too broadly. [7]

3.2.1.2 Clarity and Precision of Instructions

Another fundamental principle of prompt engineering is the creation of clear and precise instructions, as LLMs, trained on vast textual datasets from diverse sources, rely on well-structured input to generate accurate responses. When a prompt is vague or lacks detail, the model tends to produce generalized outputs, which, while broadly applicable, may not fully align with specific tasks. Conversely, structured and detailed prompts help narrow the model’s focus, reducing uncertainty and enhancing the accuracy and relevance of its responses. The more descriptive and well-organized a prompt is, the greater the likelihood of obtaining a precise, contextually appropriate answer that meets the user’s needs. The figure 3.2 shows an example of a clean and precise prompt written by Chen et al. [7]. The part of the text highlighted in green contains a clarification of the basic instructions, which allows the model to give a more desirable answer.


Model input

I want to understand the cutting edge of technology, specifically related to artificial intelligence and machine learning. Recently, artificial intelligence has become extremely popular, especially the Large Language Models which have amazed me. Please analyze the possible reasons behind the rapid advancement of this technology from at least three perspectives.

Absolutely, the rise of AI and machine learning, especially with regard to Large Language Models like GPT-4, can be seen as a convergence of several key factors. I'll analyze this from three perspectives: technological advancements, availability of data, and socio-economic demand.

1. Technological advancements: The growth of AI and machine learning models has been facilitated by several technological innovations. For starters, we've seen incredible improvements in computational power, particularly in Graphics Processing Units (GPUs) and now even more specialized AI chips. These advances have made it ...

2. Availability of data: ...

3. Socio-economic demand: ...

Figure 3.2 Example of the clear and precise prompt [7]

3.2.1.3 System message

Another widely accepted technique is incorporating a system message into the prompt. A system message serves as an extension of instructions, providing information about the role the model should assume and the expected manner of response generation. The length of a system message can vary from a short sentence to a more complex description outlining generation rules, context, or the desired output format.

3.2.1.4 Usage of delimiters

Delimiters such as triple quotation marks, XML tags, or section titles help differentiate various parts of a text, ensuring they are processed distinctly. While for simple tasks, delimiters may not significantly impact output quality, their importance increases as task complexity grows, helping to clarify instructions and reduce ambiguity. Clearly structuring prompts eliminates unnecessary cognitive load for the model, allowing it to focus on delivering precise and relevant responses. [10]

3.2.1.5 Zero-shot versus Few-shot prompting

One of the key approaches in prompt engineering is selecting between zero-shot, one-shot, and few-shot prompting, which determines whether the model requires examples to perform a given task.

Zero-shot prompting is a method where the model completes a task without any provided examples, relying solely on the knowledge acquired during pre-training [7]. This method is effective when the model is already familiar

Simple system message

You are a helpful AI assistant.

More comprehensive system message

Define model's profile and general capabilities
- Act as a [define role]
- Your job is to [insert task] about [insert topic name]
- To complete this task, you can [insert tools that the model can use and instructions to use]
- Do not perform actions that are not related to [task or topic name].

Figure 3.3 Examples of system messages [9]

with the concept or when the task is clearly defined, allowing it to generalize without additional input. Contrary to the common belief that more examples always lead to better results, Reynolds; McDonell [11] show that in some cases, zero-shot prompting can outperform few-shot prompting, especially when the prompt is well-structured and contains clear instructions.

One-shot prompting involves providing a single example before asking the model to perform the task. This technique is useful when the model needs additional contextual information but can still effectively adapt with just one example. For simple tasks or when using powerful LLMs, one-shot prompting may be sufficient, allowing the model to better align with the expected format. [7]

Few-shot prompting is a technique where the model is given multiple examples of the task within the prompt. This approach is widely used to help LLMs adapt to new tasks, improve accuracy, and ensure consistency in responses. Few-shot prompting is particularly beneficial when aiming to emulate a specific style, formatting, or reasoning process. However, research indicates that few-shot prompting does not always improve results in some cases. It is also important to understand that this technique does not train the model or alter its structure in any way [7]. It merely provides contextual information to the query. Therefore, neglecting this technique or including an excessive number of examples of the desired response can make the prompt more complex and cumbersome, potentially leading to model hallucinations and the generation of irrelevant answers.

The choice between zero-shot, one-shot, and few-shot prompting depends on task complexity, model capabilities, and the quality of the prompt formulation.

In some scenarios, adding examples may not enhance the output, whereas a well-crafted zero-shot prompt can yield more precise and relevant responses.

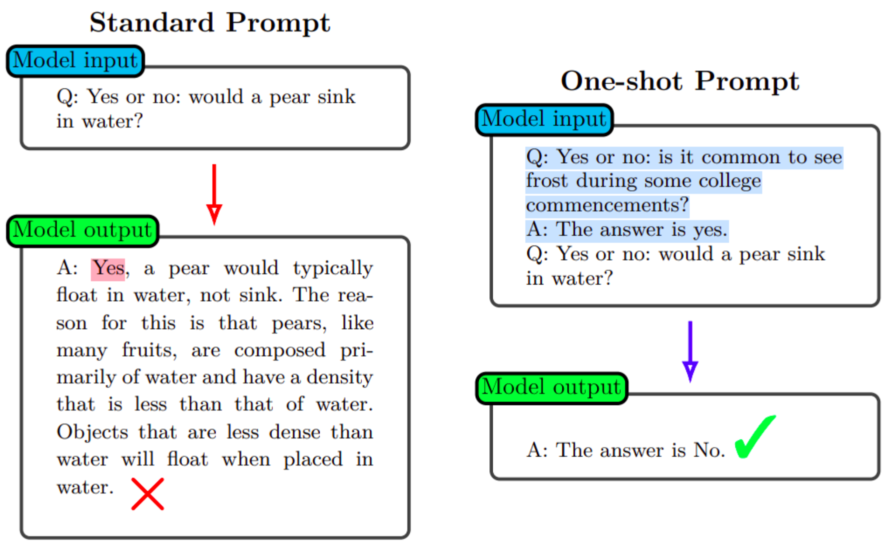

Standard Prompt
Model input
Q: Yes or no: would a pear sink in water?
Model output
A: Yes, a pear would typically float in water, not sink. The reason for this is that pears, like many fruits, are composed primarily of water and have a density that is less than that of water. Objects that are less dense than water will float when placed in water.
One-shot Prompt
Model input
Q: Yes or no: is it common to see frost during some college commencements?
A: The answer is yes.
Q: Yes or no: would a pear sink in water?
Model output
A: The answer is No.

Figure 3.4 Comparison of standard prompt and one-shot prompt [7]

3.2.2 Advanced techniques

3.2.2.1 Chain-of-Thoughts

One of the advanced techniques in prompt engineering is Chain-of-Thought (CoT) prompting, which enables large language models to break down complex problems into sequential reasoning steps before arriving at a final answer. This approach mimics the way humans think when solving multi-step problems, where intermediate steps help structure logical reasoning. [12]

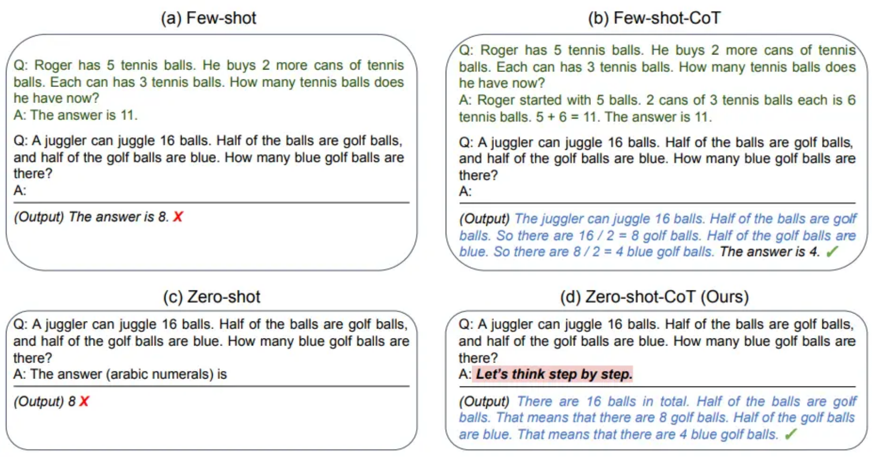
The core principle of this technique is to modify the initial prompt in a way that guides the model toward solving a complex problem by breaking it down into smaller subtasks. The model then addresses these subtasks step by step, providing reasoning and articulating its thought process throughout the solution.

CoT prompting can be implemented using both zero-shot and few-shot approaches.

- In zero-shot CoT, the model is explicitly instructed to follow a step-by-step reasoning process without any provided examples by including in prompt phrases like “Let’s solve this task step by step.” This approach relies on the model’s pre-trained ability to structure logical reasoning.
- In contrast, few-shot CoT involves supplying the model with multiple examples that demonstrate how to systematically break down complex prob-

lems into intermediate steps.

While few-shot CoT typically yields more consistent and reliable results by reinforcing structured reasoning, zero-shot CoT can still be effective, particularly in advanced models capable of implicit task decomposition. The figure 3.5 illustrates the key differences between zero-shot and few-shot CoT, highlighting their impact on the reasoning process and output quality.


(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4. ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

(Output) 8 ✗

(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Figure 3.5 Comparison of few-shot, zero-shot, and Chain-of-Thought prompting approaches [13]

According to Wei et al. [12] CoT prompting enhances reasoning in language models by:

1. Breaking down complex problems into intermediate steps, enabling additional computational resources to be allocated for solving more complex tasks.
2. Improving interpretability, allowing analysis of the model's reasoning process and easier debugging of errors.
3. Applying to any task that humans can solve through language.
4. Being easily implemented in large pre-trained models by including CoT sequence examples in the prompt.

Wei et al. [12] have conducted experiments that analyze whether a similar improvement in model performance can be achieved using alternative prompting methods (ablation study) and how sensitive the model's performance is to changes in layout structure, such as the order of examples in a few-shot setting or variations in annotation style. The study utilized the GSM8K [14] and MAWPS [15] benchmarks for evaluating math word problems. For language models, experiments were conducted using LaMDA [16] (with model sizes ranging from 422M to 137B parameters) and PaLM [12] (ranging from 8B to 540B parameters).

Ablation The following approaches were tested during the experiment.

Equation only: this approach consists of requesting the model to generate only the necessary formula before performing the calculation. Experiments have shown that this approach is ineffective for more complex tasks (GSM8K), but for simpler tasks that can be solved in 1-2 steps, it improves performance (MAWPS).

Variable compute only: this approach prompts the model to generate a sequence of dots matching the length of the required equation, isolating the impact of variable computation. Experiments indicate that this method performs about the same as the baseline, suggesting that increased computation alone is not responsible for the success of chain-of-thought prompting, but rather the benefit of reasoning through intermediate steps in natural language.

Chain of thought after answer: this method first provides the model with the answer and then shows the chain of reasoning to check if it affects the result. Experiments showed no improvements, meaning that step-by-step reasoning is needed not only for retrieving knowledge but also for the problem-solving process itself.

Robustness This experiment tested whether the writing style of chain-of-thought prompts (different annotations or permutations of examples) affects the method's effectiveness. The results showed that regardless of style and example order, chain-of-thought prompts significantly improve model performance compared to standard prompting, confirming their robustness to variations in wording and exemplars.

3.2.2.2 Generated knowledge

The figure 3.8 illustrates the core concept of the prompting method described by Liu et al. [17]. Generated Knowledge Prompting (GKP) is a technique designed to enhance logical reasoning in language models by leveraging their ability to generate relevant knowledge. Instead of relying on structured knowledge bases, GKP extracts implicit knowledge directly from the model itself before generating a response to a given question. The process consists of two main stages:

Knowledge generation: in the first stage, the language model is prompted to generate statements related to the given question. This generation process is guided by a demonstration of multiple examples, where each example

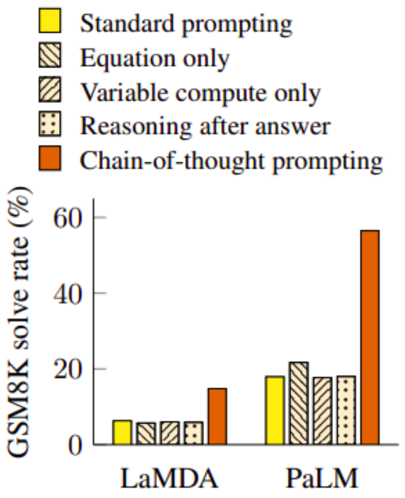


Figure 3.6 Ablation study for different variations of prompting using LaMDA 137B and PaLM 540B [12]

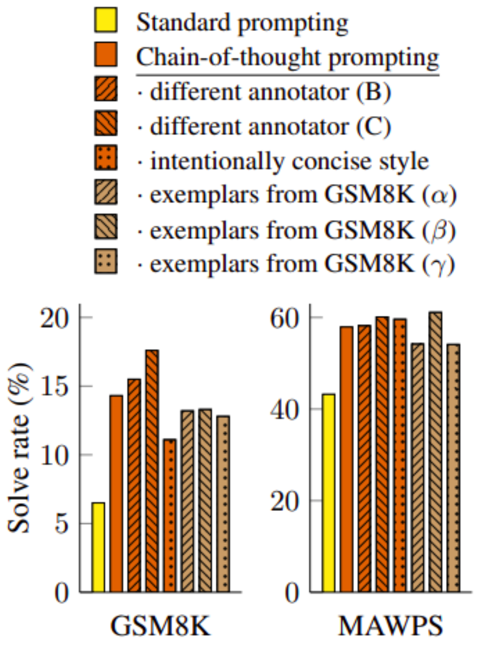


Figure 3.7 Robustness study (LaMDA 137B) for different variations of annotators and exemplars [12]

consists of a query similar in nature to the main question (which will be asked after the knowledge generation step) and a corresponding piece of knowledge that could help answer that query. These generated statements serve both as additional context that aids reasoning and as a way to transform the problem into a more structured deductive reasoning process.

Knowledge integration: in the second stage, the generated knowledge is tested to identify the most relevant information that enhances the accuracy and correctness of the response. The most suitable knowledge (either a single fact or a set of statements) is then incorporated into the main prompt containing the primary task, enabling the model to make more well-founded decisions.

Unlike traditional approaches that require direct supervision or external knowledge databases, GKP operates purely through language model prompting, making it a flexible and adaptable solution for various reasoning tasks. Experimental results demonstrate that integrating generated knowledge improves model performance across multiple commonsense reasoning benchmarks, proving that language models can function as dynamic sources of external knowledge without requiring manual intervention.

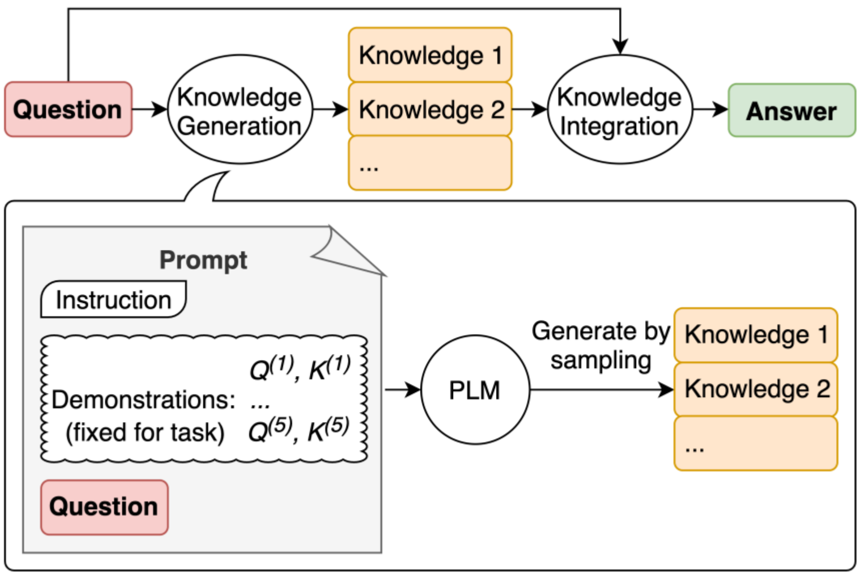


Figure 3.8 Generated knowledge prompting [17]

3.2.2.3 Meta prompting

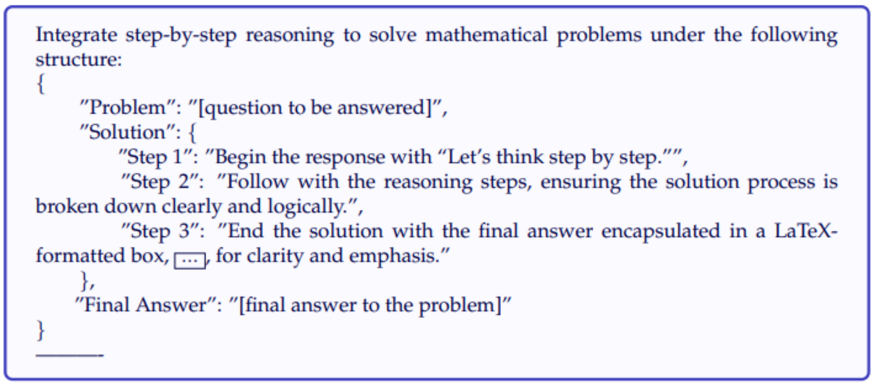
▶ **Definiton 3.1** (Meta Prompt [18]). *A Meta Prompt is an example-agnostic structured prompt designed to capture the reasoning structure of a specific category of tasks. It provides a scaffold that outlines the general approach to a problem, thereby enabling LLMs to fill in task-specific details as needed. This methodology focuses on the procedural aspects of problem-solving—the how—rather than the content-specific details—the what.*

In other words, a Meta Prompt is a universal template that helps a LLM solve tasks within a specific category without requiring explicit examples. Instead of focusing on what should be answered, it defines how the response should be structured. This approach allows the model to automatically fill in the gaps, adapting to different tasks while maintaining logical coherence in its output.

According to Zhang; Yuan; Yao [18] Meta Prompting provides two key advantages over traditional few-shot prompting methods, particularly when applied to LLMs:

Token Efficiency: since a Meta Prompt establishes a general reasoning framework rather than listing multiple examples, it significantly reduces the number of tokens used. This is especially beneficial in contexts with token limits.

Fair Comparison and Zero-Shot Efficacy: Meta Prompting can be considered a form of zero-shot prompting, as it is independent from specific ex-



```
Integrate step-by-step reasoning to solve mathematical problems under the following
structure:
{
    "Problem": "[question to be answered]",
    "Solution": {
        "Step 1": "Begin the response with "Let's think step by step."",
        "Step 2": "Follow with the reasoning steps, ensuring the solution process is
broken down clearly and logically.",
        "Step 3": "End the solution with the final answer encapsulated in a LaTeX-
formatted box, $\boxed{...}$, for clarity and emphasis."
    },
    "Final Answer": "[final answer to the problem]"
}
```

Figure 3.9 A structure meta prompt presented in JSON format [18]

amples. This makes model comparisons more objective, as LLMs approach problems without biases introduced by narrow, example-based learning.

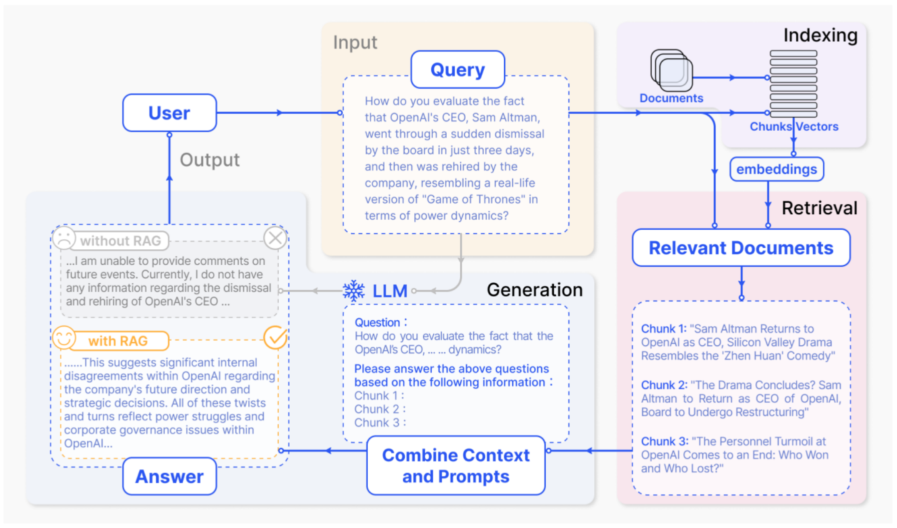
3.2.2.4 RAG

Another advanced technique in prompt engineering is the Retrieval-Augmented Generation (RAG) approach, as outlined by Gao et al. [19]. This method has gained significant attention for its ability to enhance the performance of language models by incorporating external knowledge during the generation process. RAG combines three essential stages:

Indexing: during this stage, the source data is processed and converted into a standardized text format. To accommodate the context constraints of language models, the text is divided into smaller, more manageable segments. These segments are then transformed into vector representations using an embedding model and stored in a vector database, enabling efficient similarity searches during the next retrieval phase. [19]

Retrieval: during this stage, when a user submits a query, the system converts the query into a vector representation using the same model applied during indexing. It then calculates the similarity scores between the query vector and the vectors of the text segments in the database. The most relevant segments are retrieved and used to enrich the context of the query. [19]

Generation: during this stage, the user's query and the selected documents are combined into a single prompt, which is then provided to the language model to generate a response. The response can be formulated based on the model's internal knowledge or solely using the information contained in the selected documents. [19]



How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?

...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...

......This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...

Question :
How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information :
Chunk 1 :
Chunk 2 :
Chunk 3 :

Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"

Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"

Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"

■ **Figure 3.10** An example of the RAG process [19]

An illustrative example of the described process is shown in the figure 3.10.

3.2.2.5 Reflexion

The Reflexion [20] method is an iterative self-improvement framework for LLMs that integrates structured feedback mechanisms to enhance reasoning and decision-making capabilities. It operates through three key modules:

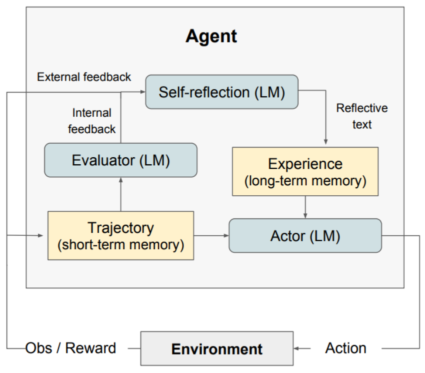
Actor: generates text and actions based on the given input and contextual observations.

Evaluator: assesses the quality of the generated response and assigns a numerical score as feedback.

Self-Reflection modul: translates the Evaluator's score into natural language feedback, providing explicit insights into errors and areas for improvement.

The process of operation of this system, indicated in the figure 3.11, can be described as follows:

1. The Actor generates an initial response based on the given prompt and available context.
2. The Evaluator analyzes this response and assigns a performance score, which quantifies the accuracy or effectiveness of the output.
3. The Self-Reflection Model transforms this numerical score and reasoning trajectory into structured verbal feedback, helping the model recognize its own mistakes and refine its approach.



Algorithm 1 Reinforcement via self-reflection

```
Initialize Actor, Evaluator, Self-Reflection:
$M_a$, $M_e$, $M_{sr}$
Initialize policy $\pi_\theta(a_i|s_i)$, $\theta = \{M_a, mem\}$
Generate initial trajectory using $\pi_\theta$
Evaluate $\tau_0$ using $M_e$
Generate initial self-reflection $sr_0$ using $M_{sr}$
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while $M_e$ not pass or $t <$ max trials do
    Generate $\tau_t = [a_0, o_0, \dots a_i, o_i]$ using $\pi_\theta$
    Evaluate $\tau_t$ using $M_e$
    Generate self-reflection $sr_t$ using $M_{sr}$
    Append $sr_t$ to $mem$
    Increment $t$
end while
return
```

Figure 3.11 Scheme and pseudocode of the Reflexion reinforcement algorithm [20]

4. This feedback is stored in the memory module, which the model references in subsequent attempts to iteratively improve its responses.
5. The process repeats until the Evaluator determines that the generated response meets the expected solution or quality standard.

A core component of Reflexion is its dual-memory system, which allows the model to retain and apply past experiences effectively:

Short-term memory: stores recent trajectories and immediate reasoning steps, enabling quick adjustments.

Long-term memory: accumulates structured self-reflections, allowing the model to recognize patterns of errors and apply strategic improvements over multiple iterations.

By leveraging verbal self-reflection, Reflexion goes beyond traditional reinforcement learning that relies solely on reward signals. Instead, it enables LLMs to internalize lessons from past mistakes, adjust reasoning pathways, and enhance problem-solving abilities over time. This approach is particularly effective for complex reasoning tasks, decision-making processes, and interactive learning environments, where a simple numerical evaluation is insufficient for meaningful learning. For these very reasons, this method has been chosen as the foundation for implementing the system presented in this work.

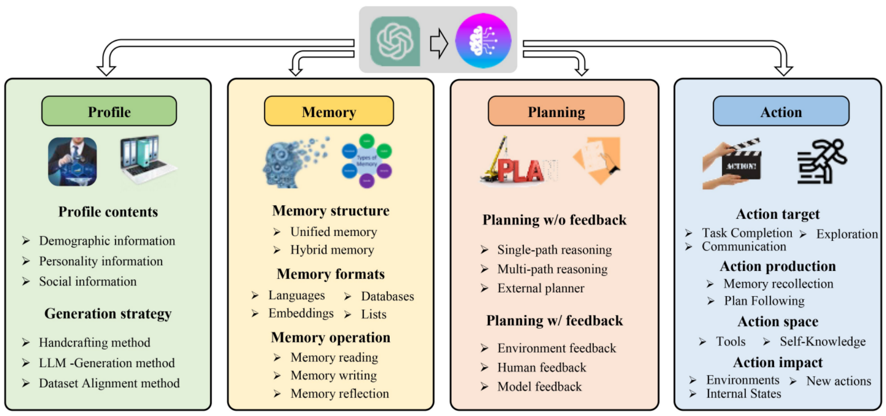
3.3 LLM-based agent

Before discussing the LLM-based agent, it is important to define what an agent is. There are many formulations of this definition. For example, Franklin; Graesser [21] formulate it as follows.

▶ **Definiton 3.2** (Autonomous agent [21]). *An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*

The LLM-based agent, according to Wang et al. [22], can be defined as an autonomous system built on large language models, which incorporates several key components in its architecture to perform complex tasks in a dynamic environment. This architecture, depicted in figure 3.12, includes the following modules:

- profiling module,
- memory module,
- planning module and
- action module.



■ **Figure 3.12** A unified framework for the architecture design of LLM-based autonomous agent [22]

Profiling module Autonomous agents carry out tasks by adopting specific roles, which are defined by the profiling module and usually embedded in prompts to influence the behavior of LLMs. These agent profiles typically contain basic information such as age, gender, and profession, along with psychological and social details that reflect their personalities. The type of profile information selected depends on the specific application. In the existing literature, three common strategies are used to create agent profil.

Handcrafting method: in this approach, agent profiles are manually crafted by defining specific traits to shape their personalities. Although this method offers flexibility, it can be labor-intensive, particularly when dealing with a large number of agents.

LLM-generation method: in this method, agent profiles are generated automatically with LLMs. The process involves setting rules for the profile structure and attributes, and optionally providing a few initial profiles as examples. LLMs are then used to create the remaining profiles. Although this approach is efficient for handling large numbers of agents, it may offer limited control over the details of the generated profiles.

Dataset alignment method: in this method, agent profiles are derived from real-world data. Information about real individuals is transformed into natural language prompts to create agent profiles. This method ensures that agent behaviors reflect real-world scenarios by accurately representing attributes like demographics. It enhances the realism of agent actions but is based on the available data.

Memory module The memory module is crucial in agent architecture, storing environmental information to guide future actions. It allows the agent to accumulate experiences, evolve, and act more consistently and effectively.

Human memory develops from sensory perception to short-term and then to long-term storage. Wang et al. [22] describe two commonly used memory structures, each combining the two memory components mentioned above in its own way.

Unified memory: this structure mimics human short-term memory through in-context learning, embedding memory information directly into prompts. This enables agents to manage and use internal states for specific tasks. However, it is limited by the context window size of LLMs, which restricts the ability to store large amounts of memory, potentially affecting performance.

Hybrid memory: this structure integrates both short-term and long-term memory systems. Short-term memory captures recent inputs, while long-term memory stores key information for extended periods. This combination enhances an agent's capacity to tackle complex tasks by utilizing both immediate data and accumulated knowledge.

The same work also describes several formats in which data can be stored in memory.

Natural languages: in this format, memory data like agent behaviors and observations are recorded in natural language. It provides flexibility and clarity, capturing rich semantic information that helps guide the agent's actions.

Embeddings: embeddings are numerical representations of data, where similar items are mapped to nearby points in a high-dimensional space. in this format, memory information is transformed into embedding vectors, improving the efficiency of memory retrieval and access.

Databases: in this format, memory information is stored in databases, allowing the agent to manipulate memories efficiently and comprehensively [22].

Structured lists: in this format, memory information is arranged in lists, making it easier and clearer to understand the meaning of the memory.

The agent interacts with the environment through three key memory operations.

Memory reading: this operation aims to retrieve valuable information from memory to improve the agent's behavior and helps agents learn from past experiences to enhance performance. It prioritizes memories based on recency, relevance, and importance, which increases their retrieval likelihood. The process of selecting information from memory can be summarized using the equation 3.3 [22].

$$m^* = \arg\min_{m \in M} \left(\alpha s^{\text{rec}}(q, m) + \beta s^{\text{rel}}(q, m) + \gamma s^{\text{imp}}(m) \right), \tag{3.3}$$

where m^* represents the selected memory based on the criteria, $m \in M$ is an element from the set of all memories M, and α, β, and γ are balancing parameters that control the weight of recency, relevance, and importance, respectively. The term $s^{\text{rec}}(q, m)$ measures the recency of memory m in relation to the query q, $s^{\text{rel}}(q, m)$ measures the relevance of memory m to the query q, and $s^{\text{imp}}(m)$ evaluates the importance of memory m. Finally, q contains the query or task that the agent is addressing.

Memory writing: this operation captures environmental data and store it in memory to improve agent performance. Key issues include avoiding redundant entries and managing limited storage. For duplicates, solutions involve merging similar data, as Achiam et al. [23], or tracking frequency counts, as Schuurmans [24]. To prevent overflow, unused memories can be deleted manually, as Hu et al. [25], or automatically replaced (oldest-first) when space runs out, as Modarressi et al. [26].

Memory reflexion: this operation enables agents to analyze their own cognitive, emotional, and behavioral patterns, similar to human introspection. This process involves synthesizing past experiences into higher-level insights. These reflections can be hierarchical, with deeper insights emerging from prior ones.

Planning module The planning module is an important component in agent design, enabling the agent to break down complex tasks into smaller, manageable subtasks. By emulating human problem-solving strategies, the planning module allows agents to think more logically and systematically, enhancing their effectiveness and reliability. According to Wang et al. [22] the planning process can be classified into two main types.

Without feedback: in this method, agents develop plans by decomposing tasks into single-step processes (using for example the Chain-of-Thought described in the Section 3.2.2.1) or multi-step processes (using for example Self-consistent Chain-of-Thought [27]). These plans are executed without any environmental influence or modification during the process.

With feedback: this method, in contrast, allows the agent to adjust its actions and strategies based on real-time feedback, which helps in refining plans and responding to unexpected changes. Autonomous agents mainly use three types of feedback to improve their decision-making: feedback from the environment, input from humans, and feedback from their own models. Environmental feedback comes from the agent's interactions with the environment, such as the results of its actions, which helps the agent adjust its plans in real-time. Human feedback provides advice to ensure the agent behaves according to user preferences and fixes any mistakes in its reasoning. Model feedback uses the agent's pre-trained systems to assess and improve its own thinking process. The incorporation of feedback is especially useful for tasks that involve long-term planning and unpredictable environments, making the agent more adaptable and capable of handling complex scenarios.

Action module The action module is a key part of autonomous agent systems, responsible for turning the agent's decisions into real actions in the environment. It works directly with the environment and is influenced by the agent's memory, plans, and role. Wang et al. [22] analyze action module from four key perspectives.

Action Goal: the actions are motivated by the goals the agent aims to achieve. These goals can include tasks like solving problems, communicating with other agents or humans, or gathering information from the environment. The actions are designed to help the agent meet these goals, which then guide the agent's behavior in the future.

Action Production: actions are generated based on different strategies. For example, the agent can use previous experiences stored in memory to decide on actions (memory recollection), or it can follow a pre-planned sequence of steps (plan following). The agent may either act based on what it has learned in the past or follow a specific plan to reach its goal.

Action Space: this refers to the range of possible actions the agent can take. These actions can include using external tools, like APIs or databases, which extend the agent's capabilities. Alternatively, the agent might rely on its internal knowledge and use its trained model to decide on the best actions based on its understanding of the task.

Action Impact: every action the agent takes has effects, both on the environment and on the agent itself. For example, actions may change the environment, like moving objects or completing tasks. They can also affect the agent's internal state, such as updating its memory or modifying its plans for future actions.

3.4 Models

This chapter provides an overview of the models used in the practical part of the work, outlining the criteria for their selection and their short description.

The main factor in choosing the models for this work was the size of their context window. The context window refers to the amount of text or information the model can process at once when making predictions or generating responses. A larger context window allows the model to handle and understand more complex input. Considering the available free options, the decision was made to use models from Google's Gemini family [28]. The Gemini family consists of advanced multimodal models developed by Google, designed to work across various data types, including image, audio, video, and text. These models are trained to perform well in each domain while also demonstrating strong generalist capabilities across multiple modalities. It should be noted that these models were accessed and utilized through API[1] calls for the purpose of this research.

Specifically, the following three models were used by Google [29] in the practical part of their work:

Gemini 2.0 Flash: this is the latest Google's multimodal model that offers next-generation features and enhanced capabilities. It can process multiple types of input, including audio, images, video, and text, and generate text-based responses. With a 1,000,000 token[2] context window, superior speed, and native tool usage, Gemini 2.0 Flash is designed for low-latency, high-performance tasks, making it ideal for powering agent-based experiences.

Gemini 2.0 Flash-Lite: this is a cost-efficient, low-latency version of the Gemini 2.0 Flash model. It supports multimodal input, including audio, images, video, and text, and provides text-based responses.

Gemini 1.5 Flash: this is a high-speed, versatile multimodal model designed to scale effectively across a wide range of tasks. It serves as a precursor to the more advanced Gemini 2.0 Flash.

Since the practical part of the work utilized the free tier of the API requests, it was necessary to adhere to the established limits that define the frequency

[1]Application programming interface.

[2]"A token is equivalent to about 4 characters for Gemini models. 100 tokens are about 60-80 English words." [29]

and size of requests to the model within a specific time period. The table 3.1 shows RPM[3], TPM[4] and RPD[5] limits.

Model	RPM	TPM	RPD
Gemini 2.0 Flash	15	1,000,000	1,500
Gemini 2.0 Flash-Lite	30	1,000,000	1,500
Gemini 1.5 Flash	15	1,000,000	1,500

Table 3.1 Rate limits [30]

3.5 Framework LangChain

"LangChain is a framework for developing applications powered by large language models (LLMs)." [31] This framework offers a wide range of tools and components that significantly simplify the development of more complex applications utilizing LLMs. Additionally, LangChain supports integration with numerous third-party products, among which one of the key integrations for this work is with Google models, which were described in Section 3.4. Further in this section, the key components of the framework that were utilized in the practical part of the work will be described.

ChatGoogleGenerativeAI: this component enables integration with Google's generative AI models. It was used to process requests and interact with the Google's language models through Google AI. Full documentation page at clickable link[6].

InMemoryRateLimiter: this component allows controlling the frequency of API requests. In the practical part of this work, the functionality of this component was used to comply with the RPM of the models described in Section 3.4. It is worth noting that this component only controls the number of requests per minute and does not control the size of the requests in any way. Therefore, the control of TPM and RPD was implemented separately. Full documentation page at clickable link[7].

PromptTemplate: this component allows the creation of various prompt templates for different queries. In this work, this component was used to create different prompts, each implementing a specific technique described in Section 3.2. Each template consists of two mandatory elements: a template, which is a text with placeholders, where variable values will later

[3]Requests per minute.
[4]Tokens per minute.
[5]Requests per day.
[6]Accessed: 17-04-2025.
[7]Accessed: 17-04-2025.

be inserted, and the variables themselves. This structure of the prompt gives it versatility and allows it to be used in different situations. Full documentation page at clickable link[8].

Chains: this not a component, but a common interface that provides the ability to create chains using a simple syntax, such as `` `prompt | llm` ``. In this work, this component is used to construct chains from prompt templates and large language models to ensure the correctness of the generated queries to the model. Once the chain is constructed, the `.invoke()` method can be called, passing a dictionary with the parameters for the prompt template. The result of calling this method will be returned as the response from the invoked model. Full documentation at clickable link[9].

FAISS: this component provides integration with FAISS[10] [32], a library developed by Facebook for efficient similarity search and clustering of high-dimensional vectors. FAISS is used for fast nearest neighbor search, making it ideal for tasks related to information retrieval and tasks where large volumes of vector data need to be processed. In this work, this component is used to implement vector stores that contain information that can be used when generating a blog. Further details about these modules are described in Chapter 4. Full documentation at clickable link[11].

HuggingFaceEmbeddings: this component allows integration with Hugging Face's pre-trained models to generate high-quality text embeddings. These embeddings are used to convert text into dense vector representations, which are essential for tasks like semantic search, information retrieval, and similarity comparison. This component enables seamless access to Hugging Face's wide range of models for embedding generation. In this work, it was used to initialize the embedding model, which was then passed into the vector store for internal operations. Full documentation at clickable link[12].

It is worth mentioning the `with_structured_output()` method, which in LangChain is used to process the model's output in a structured format. This method ensures that the results of the model's work are returned in a predefined data structure, such as a dictionary or schema. This is useful when it is necessary to extract data from textual output and convert it into the required format for further processing.

In this work, this method was applied to all the models used to ensure a consistent format for the responses, as it is necessary for the iterative processes involved in the system developed during the practical part of the work. A more

[8] Accessed: 17-04-2025.
[9] Accessed: 17-04-2025.
[10] Facebook AI Similarity Search.
[11] Accessed: 17-04-2025.
[12] Accessed: 17-04-2025.

detailed description can be found in Chapter 4. Full documentation at clickable link[13].

3.6 Related works

This section provides a brief overview and comparison of several related studies that focused on the implementation of autonomous agents based on large language models, as well as on the prediction of content popularity and audience engagement. The goal of this overview is to highlight the works that served as inspiration for the approach proposed in this thesis, by analyzing and deriving key ideas and methods from them.

3.6.1 LLM-based agents

During the implementation of the practical part of this work, the architecture of the developed system was inspired by two existing approaches: Self-Refine [33] and Reflexion [20]. These works demonstrate the potential of large language models in the context of iterative self-improvement through self-generated feedback, without the need for additional training or external supervision.

Self-Refine offers a simple and general approach in which the same language model sequentially generates an initial output, evaluates it through self-feedback, and makes corresponding improvements. This process is repeated until a satisfactory result is achieved. The main limitation of this method is that it is designed for one-off actions, without the ability to store information or learn over the long term. In other words, each new task presented to the model is solved from scratch.

In contrast, Reflexion implements a modular agent-based approach with several components: an **actor**, which performs actions; an **evaluator**, which provides feedback on those actions; a **self-reflection module**, which formulates insights in natural language; and a **memory**, where the generated insights are stored. These components interact with each other throughout the problem-solving process to achieve the given goal. A key feature of this method is the use of long-term textual memory that contains the agent's verbalized reflections, which are leveraged to improve behavior in subsequent iterations. However, in its experiments, the original paper implements memory as a simple sliding window that holds a limited number of recent experiences.

The system developed in this work can be seen as a synthesis of the two previously described approaches. From Self-Refine, it adopts the principle of iterative self-evaluation and output refinement without additional training. From Reflexion, it inherits the modular architecture and the use of memory

[13]Accessed: 17-04-2025.

for experience accumulation. A key distinction, however, lies in the implementation of long-term memory. In this work, it is realized through a vector store that mimics certain aspects of human memory. A more detailed description is provided in Chapter 4.

3.6.2 Engagement/virality prediction

Although the methodology used in this thesis differs significantly from the works described below, these studies on content popularity prediction provided valuable background and helped shape an initial understanding of how content can be evaluated.

Aldous; An; Jansen [34] propose a four-level engagement framework that categorizes user interactions from the least to the most publicly expressive: views, likes, comments or shares within the same platform, and cross-platform reposts. These levels reflect increasing degrees of user involvement and content engagement. The study analyzes the performance of news content based on several factors, including the platform where the content is published, the topic it covers, and its sentiment aspect. Proposed four-level framework was served as a partial inspiration for designing a custom engagement metric used in this thesis. More details on this metric can be found in Chapter 4.

Davoudi; An; Edall [35] presents a neural network approach for predicting dwell time[14], which reflects how interested a reader is in the content. In general, the longer the dwell time, the higher the engagement or virality of the content. The model uses both word-based and meaning-based features, including emotions, events, and mentioned people or organizations, to make its predictions.

Andariesta; Wasesa [36] explores engagement prediction for Twitter posts from major Indonesian e-commerce platforms using machine learning. The authors analyze over 12,000 tweets and identify interactivity, vividness, and timing as key predictors. Interactivity refers to the presence of elements like hashtags and links that encourage user actions. Vividness includes media types such as images and videos, which tend to attract more attention. Timing captures temporal patterns such as the day and time of posting, which can influence visibility and engagement.

[14]Expected time that users spend on an article.

Chapter 4
Implementation

4.1 Datasets

Data collection is a crucial step in solving tasks like the one addressed in this thesis. In this case, it turned out to be one of the most challenging parts of the entire project. The main objective was to construct a dataset that would include a scientific paper, a corresponding blog post explaining or summarizing the paper, and an evaluation of the blog post's quality.

Since no suitable dataset was available in open access, it had to be compiled manually. The initial data collection strategy involved searching for relevant blog posts on platforms where such content is typically published. Among the candidate sources were:

- wired.com,
- marktechpost.com,
- newscientist.com,
- medium.com and
- Google DeepMind blogs.

However, Medium emerged as the only viable source for building the dataset. It was the only platform that provided explicit user engagement metrics, such as claps and comments. These metrics were later used as a proxy for evaluating the quality of the blog posts.

After selecting the platform, the next step was to attempt automating the data collection process through web scraping. However, this approach quickly proved ineffective for several reasons. First, it was not possible to formulate clear and consistent criteria that would guarantee the relevance of a blog post, which meant that each entry would still have to be checked manually.

Second, Medium's website structure posed a technical challenge due to the inconsistency in the naming of certain HTML classes, which made reliable scraping difficult.

Moreover, in order to meet the experimental constraints and resource limits described earlier in the thesis, it was necessary to keep the dataset relatively small and well-curated. For these reasons, a decision was made to collect the dataset manually using the following search query on Google.

```
site:medium.com ("Read our paper" OR "our paper" OR
"in our paper" OR "Read our work" OR "More details
in our work" OR "Explore our research") AND ("LLM"
OR "Reinforcement learning")
```

The following part of this section provides a detailed description of each individual dataset used in this work.

Medium Blog Dataset This dataset represents the primary collection consisting of scientific articles, their corresponding blog posts, and associated evaluation scores (date of data collection: 09-04-2025). A detailed description of each basic attribute is provided below.

title_blog: the title of the blog post published on Medium.

url_blog: the direct URL of the blog post.

author_blog: the name of the blog post's author.

author_followers: the number of followers the author had on Medium at the time of data collection.

claps: the total number of claps (likes) received by the blog post at the time of data collection.

comments: the number of comments posted by readers under the blog article at the time of data collection.

publisher_followers: the number of followers of the organization that additionally published the blog at the time of data collection.

title_paper: the title of the scientific paper referenced in the blog post.

url_paper: the URL link to the original scientific paper.

publisher_blog: the name of the organization that additionally published the blog post.

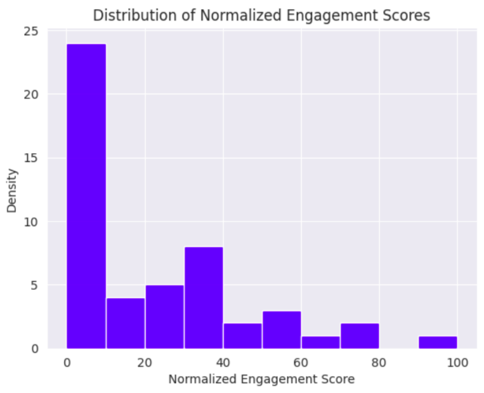


Figure 4.1 Distribution of the normalized engagement scores without logarithmization

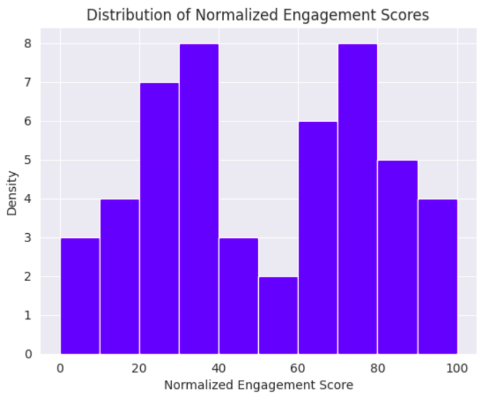


Figure 4.2 Distribution of the normalized engagement scores after logarithmization

The following attributes were added to the dataset during the data preprocessing stage:

blog_full_text: full text of the blog post.

engagement_score: engagement score for the blog post, calculated as follows:

$$\text{Engagement Score} = \frac{C + 3 \cdot M}{\log(1 + F_{\text{author}})} \cdot \frac{1}{1 + \frac{F_{\text{publisher}}}{F^{\text{max}}_{\text{publisher}}}} \tag{4.1}$$

where C is the number of claps (likes) received by the blog post, M is the number of user comments on the blog post, F_{author} denotes the number of followers the author had at the time of data collection, $F_{\text{publisher}}$ is the number of followers of the organization where the blog was published, and $F^{\text{max}}_{\text{publisher}}$ represents the maximum number of followers among all publishers in the dataset, used for normalization purposes. Thus, all available metrics are optimally taken into account when computing the blog post's engagement score.

Several alternative formulations for the blog engagement score were explored. The final selection of the presented equation was made because it leads to a more balanced distribution of engagement scores across the Medium blog dataset. The first multiplicative term incorporates the main available engagement metrics. Specifically, the number of claps C is combined with three times the number of comments M. The coefficient 3 reflects the assumption that writing a comment indicates a higher level of user engagement than giving a clap, since commenting typically requires more effort and attention. This sum is then normalized by the logarithm of the author's follower count F_{author}, which ensures comparability between authors with different audience sizes. The use of the logarithm is motivated by the observation that as the number of followers increases, the number of claps and comments grows much more slowly. Without this transformation, most scores concentrate around lower values, while a few outliers dominate the upper range.

The second part of the formula adjusts the engagement score based on the number of followers of the publisher, if one is present. When a blog is shared not only by its author but also by a publisher, such as an organization, it often gains additional exposure through the publisher's audience, which is typically much larger than the author's. To take this into account, the second multiplier returns a value between 0.5 and 1. If the blog is published only by the author, the multiplier equals 1 and does not affect the score. If a publisher with a large following is involved, the multiplier reduces the overall score. The maximum reduction is limited to 50%. This correction ensures that the engagement score reflects the actual interest in the blog content and is not overly influenced by the publisher's extended audience.

normalized_engagement_score: normalized blog post engagement score in the range from 0 to 100. The main goal was to compute the engagement scores in a way that ensures a relatively balanced distribution across the entire dataset. As a result, it was decided to apply a logarithmic transformation to the value obtained after using the formula 4.1. The figure 4.1 shows the distribution of blog scores after applying the formula 4.1, while figure 4.2 shows the distribution of the same scores after applying a logarithmic transformation. Thus, the final version of the blog score computation can be expressed as follows.

$$\text{Engagement Score} = \log\left(\frac{C + 3 \cdot M}{\log(1 + F_{\text{author}})} \cdot \frac{1}{1 + \frac{F_{\text{publisher}}}{F^{\max}_{\text{publisher}}}}\right) \tag{4.2}$$

engagement_level: a categorical attribute that reflects a qualitative assessment of user engagement with a blog post. It is derived from the normalized engagement score using predefined thresholds. The possible values of this attribute and their corresponding numerical intervals are as follows:

- *Excellent*: score ≥ 80,
- *Very Good*: $65 \leq$ score < 80,
- *Good*: $35 \leq$ score < 65,
- *Average*: $20 \leq$ score < 35,
- *Bad*: score < 20.

The figure 4.3 shows the distribution of the blog engagement levels after applying mentioned mapping.

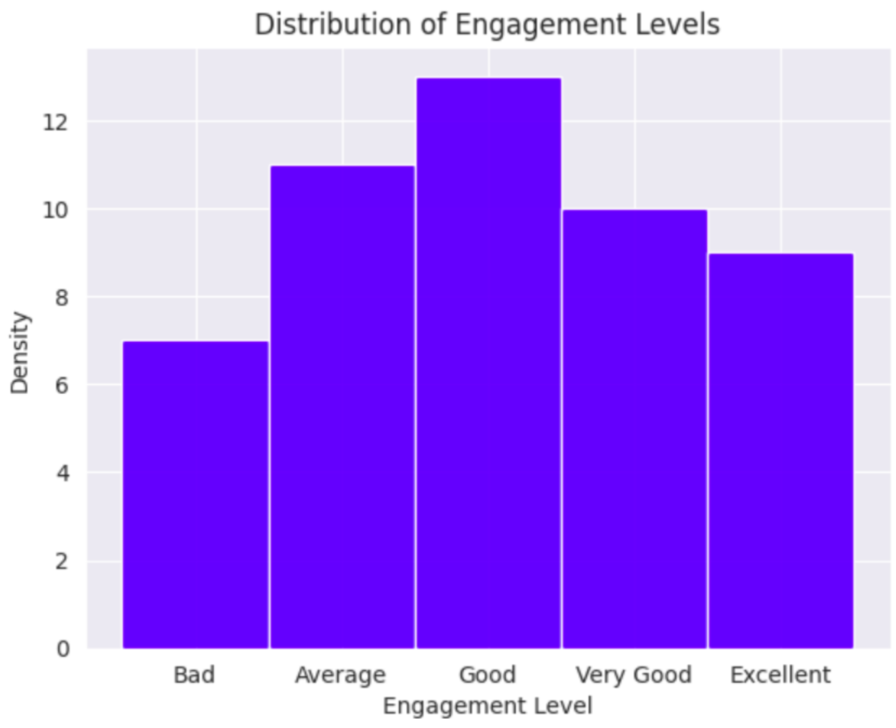


Figure 4.3 Distribution of the engagement levels

Google Deepmind Blog Dataset This dataset was also manually collected for conducting additional experiments (date of data collection: 10-04-2025). A brief description of each attribute is provided below.

id: unique identifier for the blog post entry in the dataset.

url_blog: direct URL link to the blog post.

blog_full_text: the complete textual content of the blog post.

title_blog: the title of the blog post.

NeurIPS Papers This dataset, obtained from Kaggle, contains metadata of research papers presented at the NeurIPS[1] conference from 1987 to 2019. The dataset was used to conduct experiments with the full pipeline, particularly for evaluating blog generation models on longer, scientific input texts.

During preprocessing, all papers lacking an abstract or full text were removed, and only those with full texts exceeding a minimum length (defined as a combination of character count per page and a threshold page number) were retained. This ensured that only complete and sufficiently long documents were included. The resulting subset was sorted by the size of the full text and

[1]Conference on Neural Information Processing Systems

publication year to prioritize more recent and substantive content. Irrelevant columns such as `source_id` and `year` were dropped, and column names were changed to align with other datasets used in the thesis (e.g., `full_text` renamed to `paper_full_text`, `title` to `title_paper`). Finally, the dataset was limited to the top 74 entries (the nearest even number to the actual number of samples retained after preprocessing). Thus, the dataset retains the following attributes.

title_paper: title of the research paper.

abstract: abstract summarizing the paper content.

paper_full_text: full body text of the paper.

full_text_size: character count of the full text.

4.2 Architecture

The system is designed as a modular and extensible pipeline for generating engaging blog posts from scientific papers. Its core principle lies in iterative self-improvement, achieved through a feedback loop that combines self-reflection with the accumulated knowledge stored in long-term memory. By analyzing previously generated outputs and leveraging insights from similar past cases, the system continuously refines its performance.

At the heart of the architecture is the `BlogGenerator` class, which coordinates the entire process of content generation, evaluation, and memory management. It integrates a vector store for retrieving contextually relevant examples, a dual-model setup for generation and evaluation, and a configurable retry mechanism that enables output refinement across multiple iterations. The components of this class are described in more detail below.

Input Processing: the system accepts either a direct text of the scientific paper or a URL, from which the full text is extracted.

Vector Store Retrieval: to support the RAG paradigm, the system employs a vector store, which contains vector representations of all scientific papers and their corresponding blog posts from the dataset (with Medium blogs) described in the previous section. Importantly, only those papers whose associated blogs have received an evaluation of either "Very Good" or "Excellent" are stored. This design choice ensures that the generation model is guided from the outset by high-quality examples, increasing the likelihood of producing similarly engaging outputs.

Content Generation: the core generation task is performed by a large language model (Gemini 2.0 Flash described in Section 3.4), configured to produce structured blog content. The generation process is driven by prompts that vary depending on the iteration:

- **Initial Prompt:** this prompt is used during the first iteration of generation and follows the RAG approach described in Section 3.2.2.4. A pair consisting of the most semantically similar scientific paper and its corresponding blog post is retrieved from the vector store described above and injected into the prompt as an example. This helps the model understand the expected format, tone, and depth of the output from the very beginning.
- **Retry Prompts:** used in subsequent iterations, these prompts include structured feedback received from the evaluation model in the form of suggested improvements (reflexion). In addition to the list of improvements, the prompt also incorporates the full text of the previously generated blog and the evaluation score provided by the evaluator model. This comprehensive context allows the generator to better understand what aspects need improvement and to iteratively refine the output. If the long-term memory module is enabled, additional context from the most relevant previously generated blog stored in memory is also included. This helps reinforce learning from past examples that were successfully evaluated. Both the self-reflection mechanism and the memory module can be disabled by the user. In that case, the system will rely solely on the RAG approach, re-generating the blog from scratch in each iteration using the same static example, until either a satisfactory result is produced or the maximum number of allowed attempts is reached.

Content Evaluation: the quality of the generated blog post is assessed by a separate evaluation model, also based on Gemini 2.0 Flash. This model is used specifically for automated evaluation and is responsible for classifying the blog post according to engagement level while also providing structured feedback, including specific suggestions for improvement. These outputs are then used in the next generation attempt, forming a feedback loop that enables the system to iteratively enhance the output quality. All references to evaluation in this work refer exclusively to this LLM-based automated process. Following a series of experiments (as described in the corresponding chapter), the Chain-of-Thought (CoT) prompt was selected as the primary prompt for the evaluation model. The explanation and justification for choosing this approach, as well as a comparison with alternative prompting strategies, are presented in the experiments in Chapter 5.

Memory: the long-term memory component (`LongTermMemory`) is responsible for storing all previously generated blog posts, regardless of their evaluation outcome, along with corresponding metadata such as engagement scores and suggested improvements. This memory is persistent across sessions and acts as a dynamic knowledge base of both successful and less successful generations. During the generation process, especially in retry iterations, the system can query the memory to retrieve the most relevant past blog based on semantic similarity to the current draft. The retrieved blog, along with its quality score and evaluator feedback, is then incorporated into the prompt. This allows the generator to learn not only from examples of excellence, but also from past mistakes. A more detailed description of how this module operates is provided in Section 4.3.

Algorithm 1: Blog generation pseudocode

```
   Input: Scientific paper (text or URL)
   Output: Generated blog post
 1 Initialize vector store and modules
 2 Initialize long-term memory
 3 Find similar example paper and blog
 4 for each generation attempt (max N) :
 5     if first attempt or (reflexion is disabled and memory is disabled) :
 6         Generate blog using RAG prompt
 7     else:
 8         if memory is enabled :
 9             Retrieve similar blog from memory
10             Generate blog using retry prompt with memory
11         else:
12             Generate blog using retry prompt with reflexion
13     Evaluate generated blog
14     Add pair of the blog and its evaluation to the long-term memory
15     if evaluation == "Very Good" or evaluation == "Excellent" :
16         Save updated long-term memory module
17         Save blog to file
18         return blog
19     Update best blog if current is better
20 Save updated long-term memory module
21 Save best blog to file
22 return best blog
```

Thus, the blog generation process, as illustrated in the pseudocode 1, proceeds iteratively with the goal of producing the highest quality output. First, the vector store and supporting modules, including the long-term memory, are initialized. Then, the system retrieves a similar example paper and its corresponding blog post. If it is the first attempt, or if both self-reflection mechanism and memory are disabled, the blog is generated using a basic RAG prompt. On subsequent attempts, the process branches depending on the available capabilities. If memory is enabled, a relevant blog from memory is retrieved and used within a retry prompt. Otherwise, reflexion techniques are applied to guide the regeneration. Each generated blog is then evaluated and stored in long-term memory along with its evaluation. If the evaluation is "Very Good" or "Excellent", the blog is saved and returned. Otherwise, the algorithm continues to refine the output, always keeping track of the best version, which is saved at the end. By default, the process is repeated up to five times.

4.3 Implementation details

This chapter provides a more detailed description of selected components that play an important role in the logic and operation of the system. The goal is to explain how these parts work and why they are designed in a specific way, in order to give a better understanding of how the system works.

The complete source code implementing these components is included in the attachment to this thesis. The structure of the attachment is described in Chapter B, and all source code files can be found in the `src/` directory.

Long-Term Memory Module In the first step, the module attempts to load an existing FAISS vector store from the path specified in the configuration. If loading fails (e.g., the store does not yet exist), a new vector store is initialized with dummy data to prevent runtime errors related to empty indices. The class supports both raw embedding generation via a sentence-level encoder and compatibility with LangChain's FAISS wrapper for vector indexing and search. Embeddings are generated using the `all-MiniLM-L6-v2` model, a lightweight and efficient transformer-based model well-suited for semantic similarity tasks.

When a new blog post is generated, the system attempts to add it to the long-term memory using the `add_to_memory` method. This process begins by encoding the blog content into a vector representation. The module then performs a similarity search against existing entries to determine whether the blog is sufficiently novel. If two similar entries are found and both fall within a predefined distance threshold, the system performs an upsert[2] operation. The distance threshold was determined empirically: the model generated ten blog posts for the same input query, and pairwise distances were calculated between all resulting vectors. The maximum observed distance among these was rounded to the nearest tenth and subsequently used as the threshold for similarity filtering.

The upsert logic ensures that memory only retains the best available version among similar blog entries. Specifically, when a new entry is to be inserted into the memory store, the system first searches for the two most similar existing entries by computing vector distances. If the distances between the new entry and both of the nearest neighbors are below the previously defined threshold, the system compares all three entries: the new one and the two existing ones.

[2]The upsert operation controls for duplicates during insertion. If a similar entry already exists, it is replaced with the new one; otherwise, the new item is simply added to the database

It then retains the entry with the highest evaluation score and removes the other two from the store.

If no similar entries are found or their similarity scores exceed the threshold, the new blog is simply added to memory without further modification. This mechanism balances memory efficiency with the goal of preserving mainly high-quality examples for future reuse. The implementation of this logic is located in the `src/long_term_memory.py` file, as part of the attached source code.

Models Setup The blog generation and evaluation components are configured using the Gemini 2.0 Flash model, which is applied to both tasks with a temperature parameter set to 1. The generator is initialized with a structured output schema defined by the `BlogGeneration` class, ensuring that the output includes the complete blog text along with the title. Similarly, the evaluator is configured with the `BlogClassificationCoT` schema, which captures a detailed evaluation of the blog through a chain-of-thought reasoning process. This evaluation includes a qualitative rating, the reasoning behind the assessment, and suggestions for potential improvements. Both generator and evaluator are set to return not only structured outputs but also raw model responses for further control of the token usage. By default, the system is configured to allow up to three attempts to obtain a valid response from the model. In cases where the model fails to produce a response, it returns null values, which prevent the system from proceeding with subsequent operations. The configuration of the models is located in the `src/models_setup.py` file, as part of the attached source code.

Handling Rate Limits The system enforces API usage limits by continuously monitoring the number of requests and tokens consumed. It supports three types of constraints: requests per day (RPD), requests per minute (RPM), and tokens per minute (TPM). If the daily limit is exceeded, the process is halted with an error. For the RPM and TPM constraints, the system automatically pauses execution and waits until the beginning of the next minute before resuming, ensuring compliance without manual intervention. This behavior is managed using internal counters and timestamps. The implementation of this logic is located in the `src/blog_generator.py` file, as part of the attached source code.

Text Extraction The blog texts are extracted directly from web pages (Medium or Google DeepMind) using the Selenium automation framework. The system loads the target URL in headless mode using a Chrome browser. From the main `<article>` HTML element, only relevant content tags are selected, such as headings (`<h1>`, `<h2>`, `<h3>`), paragraphs (`<p>`), and list items (`<li>`). Irrelevant elements are filtered out using predefined HTML classes or specific phrases related to subscription or author/publisher mentions.

Scientific papers in PDF format are processed by first downloading the file using the `requests` library. The PDF is then opened and parsed with the `PyMuPDF` library (`fitz`), which extracts the text content from each page and combines them into a single string for further processing. The implementation of this logic is located in the `src/text_extraction.py` file, as part of the attached source code.

User Interaction The user interaction with the system is implemented via a command-line interface, where users are sequentially asked to provide input parameters for blog generation. The process begins with setting configuration values, such as the maximum number of blog generation attempts and the number of retry attempts for valid model responses. Users are then asked whether they wish to enable the self-reflection mechanism and long-term memory module.

Once configuration is complete, users specify the format of their scientific paper, either by providing a direct URL to an `arXiv` PDF or by supplying a local file containing the full text. Based on the user's input, the appropriate data extraction method is triggered, and the system proceeds to generate a blog using the `BlogGenerator` module. If the generation is successful, the resulting blog post is saved. The location of the final blog output is indicated in the program logs. The implementation of this logic is located in the `src/main.py` file, as part of the attached source code.

Chapter 5

Experiments and results

This chapter provides a chronological overview of the key experiments conducted throughout the development of the system. The focus is first placed on the evaluation module responsible for assessing the quality of the generated blog posts. Following that, the chapter details experiments carried out on the overall system to analyze its performance and effectiveness.

5.1 Blog Evaluation

5.1.1 Data Preparation

The experiments evaluating the blog quality assessment module were conducted using two datasets, both of which were described earlier in Section 4.1. Specifically, the primary dataset consisted of Medium blog articles, while the Google DeepMind blog dataset was employed to support additional experiments. Both datasets were loaded and reindexed to ensure a consistent internal structure for further processing.

The Medium blog dataset was then split into validation and test sets in a 60/40 ratio, respectively. The validation set was used to identify the most effective combination of model and prompt, while the test set served to evaluate the expected performance of the selected configuration on unseen data. It was initially assumed that the results on both sets would be consistent; however, this assumption did not hold true, and the differences are discussed in detail later in this section. In both subsets, the target variables were extracted and handled separately. These included the normalized engagement score used for regression tasks and the engagement level used for classification.

In addition, a curated set of blog examples representing distinct engagement levels was loaded to support few-shot prompting during experiments. These examples were carefully selected from the primary Medium dataset and removed from the dataset with Medium blog articles prior to splitting. This step was crucial to ensure that none of the prompt examples appeared in either the validation or test sets, thereby preserving the integrity and fairness of the experimental evaluation.

5.1.2 Prompt Creation

The subsequent stage involved the gradual manual development and refinement of prompts. This process took a considerable amount of time. The main goal was to achieve the desired result through careful prompt adjustments. The main prompts are included in the appendix, and the full set of prompts along with sample model responses can be found in the attached files. Including all of them in the main text would take up too much space.

The process started with a very simple prompt consisting of a single sentence.

Simple prompt

```
Rate this blog post in one word.
{blog_text}
```

Example of the output

```
Informative
```

Over time, the prompt was made more complex by adding new elements and techniques, which are described in detail in Section 3.2.

After a long process of enriching and modifying the prompt, the first candidate for the main prompt used in the evaluator module was the **two-shot prompt**, which is provided in Appendix A.4. It consists of the following parts:

Profile message: a brief description of the task and the context in which the model operates.

Task description: the core assignment given to the model. This part is similar across all prompts and asks the model to evaluate a blog post based on several criteria derived from Chapter 2.

Clarifications: instructions to help avoid typical mistakes made by the model. For example, it is specified that visual and interactive elements should not be considered when evaluating the blog because the content was extracted without them.

Expected output: the required format of the model's response, which includes a numerical score from 1 to 100 and suggested improvements presented in JSON format.

Examples: two sample responses with different scores to guide the model toward the expected type of answer.

Blog: the blog post to be evaluated.

All other prompts created for evaluating blog quality share a similar structure, with only minor differences depending on the technique being applied. Therefore, the text below focuses solely on these differences without repeating the full structure.

Five-shot prompt: in this prompt, the model is not asked to score the blog on a scale from 1 to 100. Instead, it must classify the blog using one of five predefined categories. The prompt includes five example blogs, each labeled with a different classification to guide the model.

Zero-shot Chain-of-Thought prompt: this version, which is provided in Appendix A.5, does not include any examples. Instead, it instructs the model to perform step-by-step reasoning before providing the final classification.

Generated Knowledge prompt: this prompt also contains no examples. Before delivering the final evaluation, the model is required to generate general background knowledge that helps extend the context and potentially improves the accuracy of the blog classification.

Meta prompt: this version, which is provided in Appendix A.6, shares only three components with the **two-shot prompt**: the profile message, clarification notes, and the blog itself. The fourth component is unique to this type and includes a detailed list of instructions the model should follow. These steps end with generating a classification and suggesting potential improvements to the blog.

5.1.3 Performance Evaluation

This section presents the experimental results obtained by evaluating the scoring models across several datasets, including a validation set, a test set, and a dataset of blogs from Google DeepMind. The evaluation focuses on comparing three models described in the corresponding Section 3.4.

Since generation of the blog evaluation by a language model is inherently stochastic, each experiment was repeated five times to compute the mean and the 95% confidence interval 5.1 [37], which were then used for further analysis.

$$\left[\bar{x} - t_{\alpha/2,\,n-1} \cdot \frac{s}{\sqrt{n}},\, \bar{x} + t_{\alpha/2,\,n-1} \cdot \frac{s}{\sqrt{n}}\right], \tag{5.1}$$

where $\bar{x}$ is the sample mean, s is the sample standard deviation, and n is the number of observations. The term $t_{\alpha/2,\,n-1}$ represents the critical value from the Student's t-distribution with $n-1$ degrees of freedom. The value of α is calculated as $1 -$ confidence; for example, for a 95% confidence level,

$\alpha = 0.05$. This interval estimates the range within which the true population mean is likely to fall with a specified level of confidence.

For prompts that require categorical classification rather than numerical scoring, the following mapping was applied to convert qualitative ratings into numerical values:

- `Excellent` $\rightarrow 5$
- `Very Good` $\rightarrow 4$
- `Good` $\rightarrow 3$
- `Average` $\rightarrow 2$
- `Bad` $\rightarrow 1$

Model performance was assessed using the following classic metrics:

Root Mean Square Error (RMSE): this metric was used because it measures the square root of the average of the squared differences between predicted and actual values, making it particularly sensitive to larger errors, which should be avoided in the task at hand.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{5.2}$$

where y_i denotes the actual (true) value for the i-th observation, while $\hat{y}_i$ is the corresponding predicted value produced by the model. The symbol n represents the total number of observations.

Mean Absolute Error (MAE): this metric is easy to interpret as it represents the average magnitude of prediction errors.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{5.3}$$

where y_i denotes the actual (true) value for the i-th observation, while $\hat{y}_i$ is the corresponding predicted value produced by the model. The symbol n represents the total number of observations.

Accuracy (for classification tasks): this metric was used when the model was tasked with assigning blogs to predefined quality categories.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{5.4}$$

Validation Set The initial experiments were conducted using a **two-shot prompt** that required the model to produce a numerical score ranging from 1 to 100. The corresponding figs. B.1 and B.2 and table 5.1 present visualizations of the results.

Model	RMSE	MAE
Gemini 1.5 Flash	44.42	37.58
Gemini 2.0 Flash	43.63	36.68
Gemini 2.0 Flash-Lite	41.96	35.20

Table 5.1 Average error values (two-shot prompt)

Based on the presented results, it can be concluded that the **Gemini 2.0 Flash-Lite** model outperforms the other evaluated models in terms of both RMSE and MAE values. However, it is important to note that the overall average error values (both absolute and squared) are relatively high. This suggests that the current prompt does not yield satisfactory results and is not well-suited for accurate numerical evaluation of blog content.

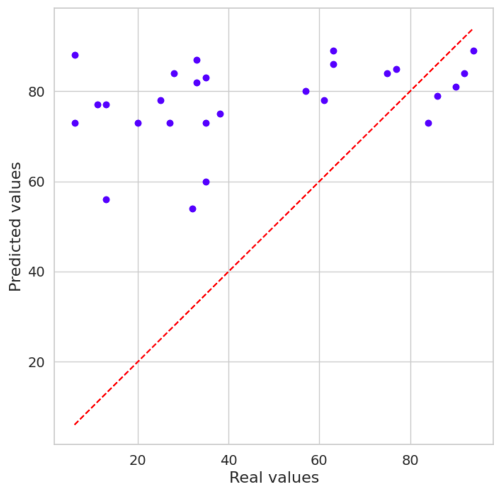


Figure 5.1 Real vs. average predicted values across 5 experiments (Gemini 2.0 Flash-Lite + two-shot prompt)

This observation is further supported by the scatter plot in figure 5.1, which visualizes the relationship between the predicted and actual values. Ideally, the data points should align closely along the diagonal line, indicating accurate predictions. However, in this case, most of the points deviate significantly from the diagonal, indicating a lack of predictive precision.

It is also evident that the predicted values mostly fall between 50 and 90. To potentially improve model performance, the predictions were subsequently normalized to a range between 0 and 100.

Based on the results shown in figs. B.3, B.4 and 5.2 and table 5.2, normal-

Model	RMSE	MAE
Gemini 1.5 Flash	42.94	36.02
Gemini 2.0 Flash	31.18	24.84
Gemini 2.0 Flash-Lite	39.76	32.90

Table 5.2 Average error values after normalization (two-shot prompt)

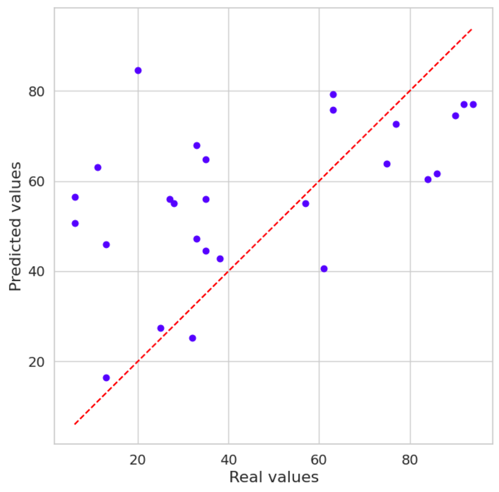


Figure 5.2 Real vs. average predicted values across 5 experiments after normalization (Gemini 2.0 Flash + two-shot prompt)

ization seems to have improved the performance. After applying it, the Gemini 2.0 Flash model achieved the best results compared to the others. However, the mean absolute error is still quite high, around 25, which is too much to consider the tested prompt effective for this task.

At this stage, it was evident that the tested models are unable to accurately predict blog quality using numerical ratings, with the best MAE being around 25 units, which represents a quarter of the entire rating range. As a result, a simpler and more suitable approach was tested, which involved classifying blog quality based on the predefined set of ratings ranging from "Bad" to "Excellent". The results presented in figs. B.5 to B.8 indicate that the combination of the **Gemini 2.0 Flash** model and the **Meta prompt** outperforms the other evaluated configurations. The mean absolute error is 1.0, which corresponds to a one-point error. For example, a blog rated as "Good" might be classified as "Average" or "Very Good", and the mean squared error is 1.32. However, the prediction accuracy values shown in figs. B.9 and B.10 are quite low, suggesting that the selected models, in combination with the tested prompts, may not be capable of providing a fully reliable classification of blog quality.

The set of scatter plots shown in fig. 5.3 demonstrates that the classification approach suffers from the same issue as the numerical rating. In most cases, the model assigns a rating of 3, which is equivalent to "Good". Therefore, it can be assumed that the relatively low RMSE and MAE values are due to random circumstances, including the fact that a significant number of blogs in the dataset are rated as “Good”, “Average” or “Very Good”.

Nevertheless, these prompts are retained in the subsequent stages of the system, as the evaluation model not only assigns a quality label but also generates specific suggestions for improvement. These suggestions are then used to iteratively refine the blog post across multiple generations, which can lead to overall quality enhancement over time. To examine these assumptions, the remaining experiments presented in this work were conducted.

In all the other experiments described below, only the Gemini 2.0 Flash model was used, as it outperformed the other models in almost every case in the previously described experiments.

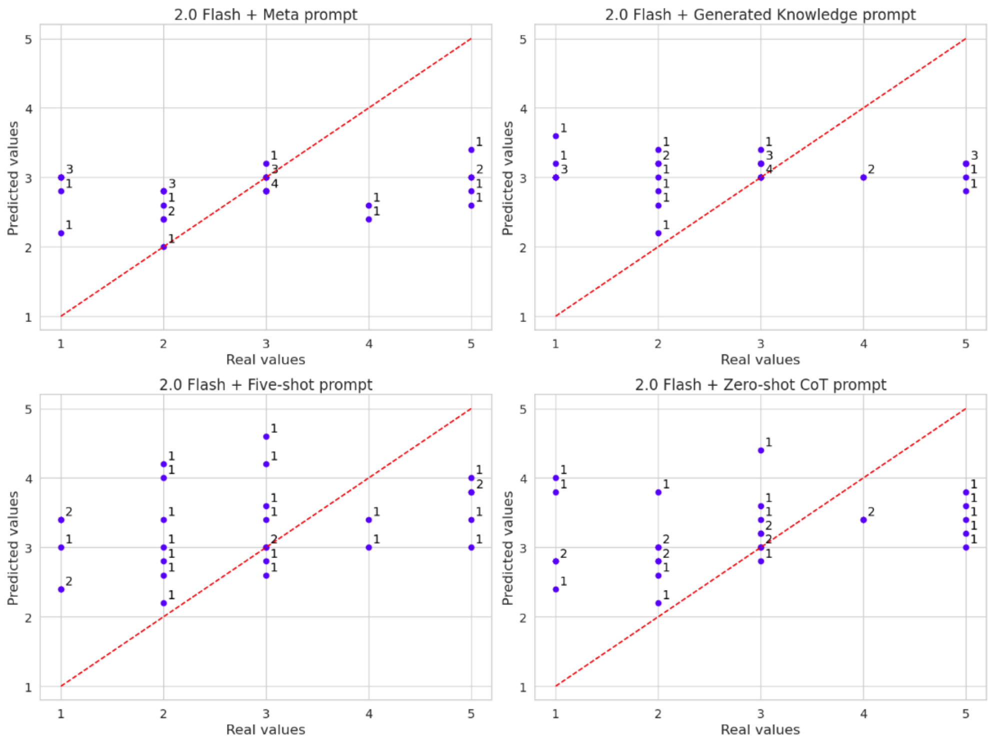


Figure 5.3 Scatter plots comparing real (from the validation set) and predicted blog ratings using different prompts and the Gemini 2.0 Flash model, each point is labeled with a number indicating how many data points overlap at that location, the values shown are averaged over five experimental runs

Google Deepmind Blogs The experiment on this dataset was conducted to strengthen the confidence in the statement made at the end of the previous paragraph, specifically that the model tends to classify blogs as “Good” in most cases. The blogs in this dataset do not have predefined ratings. However, since all the blogs in this collection are published on the Google DeepMind platform, it is assumed that the distribution of blog ratings will be shifted toward the higher end (i.e., a higher likelihood of the blog receiving a top rating). This assumption is based on the fact that all blogs are written by a carefully selected team of experts at Google, whose work is generally considered to be of high quality.

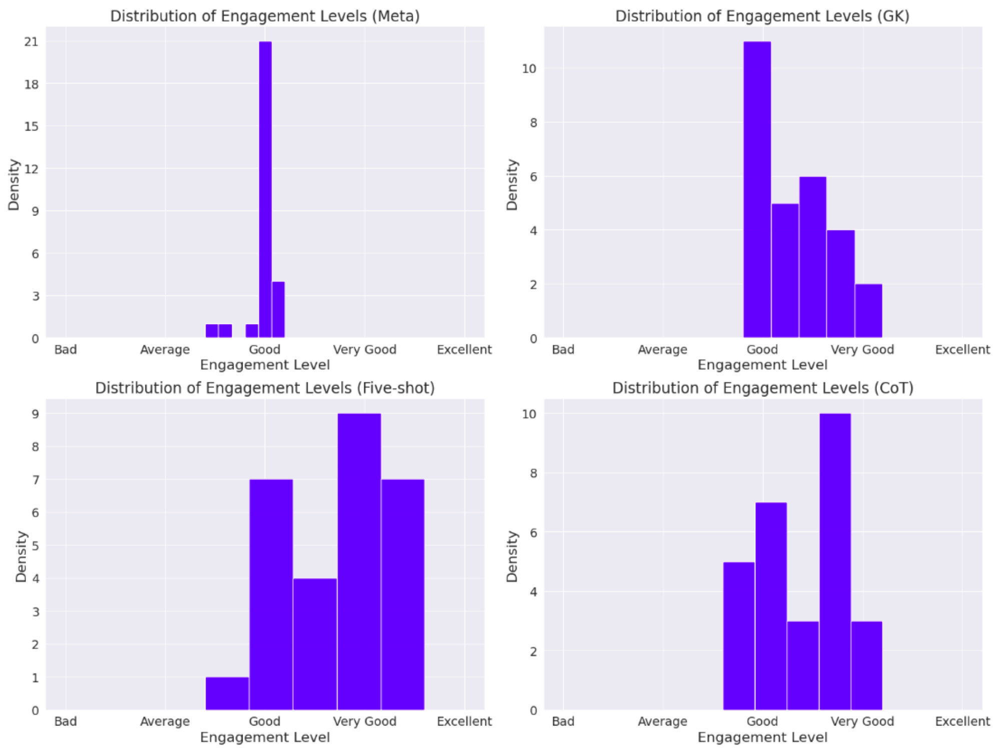


Figure 5.4 Distribution of averaged engagement levels for Google DeepMind blogs (Gemini 2.0 Flash)

The experiment compared the ratings of the Gemini 2.0 Flash model using different prompts. From the results shown in fig. 5.4, it is evident that the combination of Gemini 2.0 Flash and the Meta prompt, which performed the best in the previous validation experiment, classifies nearly all (21/28) Google blogs as “Good”. In contrast, using other prompts results in more varied ratings.

Test Set Before conducting experiments on this dataset, it was assumed that the results would be similar to those observed on the validation set. However, the results shown in figs. B.11 to B.13 and 5.5 turned out to be the complete opposite of those obtained on the validation set. In this case, the combination of the Gemini 2.0 Flash model and the Meta prompt yielded the worst performance to the others. This highlights the fact that the performance of the entire evaluation module is less dependent on the predictive capabilities

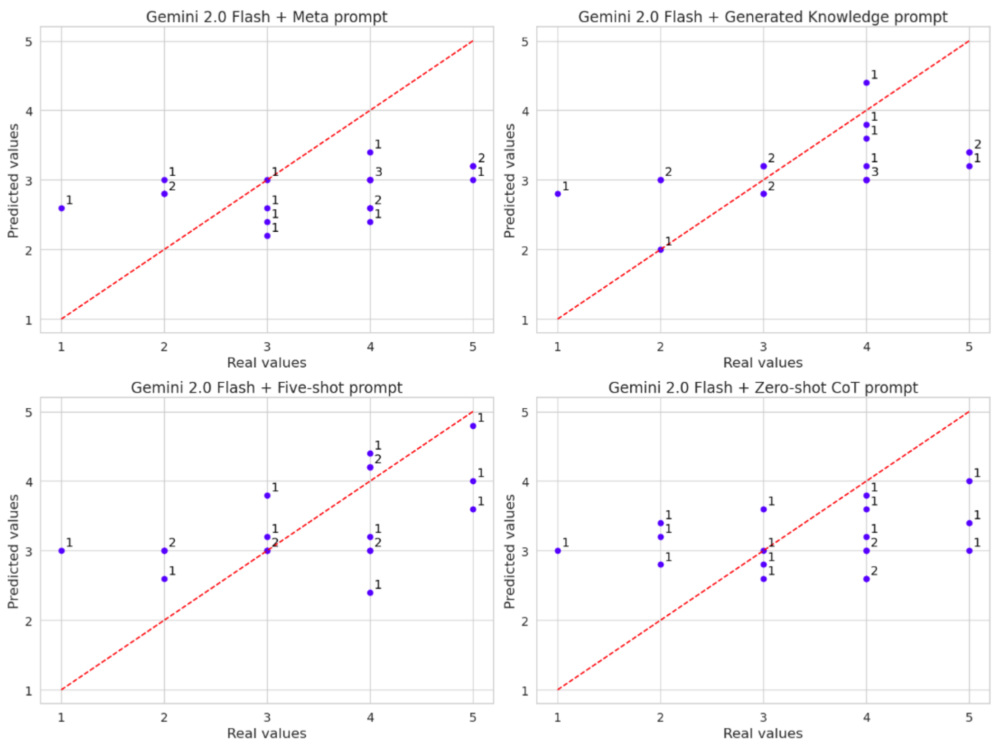


Figure 5.5 Scatter plots comparing real (from the test set) and predicted blog ratings using different prompts and the Gemini 2.0 Flash model, each point is labeled with a number indicating how many data points overlap at that location, the values shown are averaged over five experimental runs

of the used LLMs itself and more on the dataset used for the experiment. In simpler terms, the model's performance is largely influenced by factors such as the random seed used during experiment.

General Conclusion After the Experiments Based on the collected and derived data, the results of the conducted experiments revealed that none of the combinations of tested models and prompts were able to adequately predict blog post quality ratings. Several factors could explain this outcome.

First, the custom metric used in the dataset, which is designed to represent a blog's rating based on Medium blog data, may not align with the actual quality assessment of a blog. This is because the true rating depends on a much broader range of factors (many of which are unknown), beyond just the number of likes, comments, and followers of the author or publisher. However, even the custom metric introduced in this work enables comparative analysis of the generated content when used consistently. This capability is undoubtedly an essential aspect of the content generation process.

Second, the dataset only includes the textual content of the blogs, without considering visual and interactive elements that significantly impact the overall quality of a blog post. These missing components might play a crucial role in how blogs are rated in practice.

Lastly, the pre-trained models used in these experiments were trained on vast amounts of data, yet they were not fine-tuned or adjusted during this study. The models remained static, which may have limited their ability to adapt to the specific nature of the experiment. Although fine-tuning was not a primary method used in this work, it represents a potential avenue for improving the evaluation model's performance in future. Additionally, the experiments were conducted on a relatively small dataset due to limitations on the number and size of queries that could be made to the models within a given period of time.

However, in all cases, the model not only generated a specific blog rating but also suggested possible improvements that could be useful for the iterative enhancement of the blog. Therefore, at this stage, the decision was made to fix the combination of the **Gemini 2.0 Flash** model and the **Chain-of-Thought prompt** as the evaluation module for experiments on the entire system that generates blogs. Given the lack of success in the previous experiments, this choice is justified mainly by several practical considerations. First, This combination demonstrated sufficient variability in blog post assessments, which made it possible to monitor improvements across iterations. Second, this combination relatively "cost-effective" in terms of token usage (TPM). The decision to fix this combination of the model and prompt was also influenced by the fact that, at the time of these experiments, there was no more time available for making significant changes to the dataset or the evaluation module that could have positively impacted the results.

5.2 Blog Generation

Data Preparation The primary dataset containing blogs from Medium was not used directly in these experiments because it was employed exclusively for populating the vector store. The vector store provided additional contextual information for the first iteration of blog generation within the RAG approach. Instead, a separate dataset with NeurIPS papers, described in Section 4.1, was used as the input for blog generation during testing.

Before the experiments, the dataset of scientific articles was split into two parts, each containing 37 elements. The first part was used for experiments on the system without the long-term memory module. In these experiments, the system's memory was always initialized as empty. During testing, both in the RAG-only configuration and with the self-reflection mechanism enabled, the memory was gradually populated with generated blogs and their associated metadata, including quality assessments produced by the evaluation module.

The second part of the dataset was reserved for experiments with the long-term memory module activated. This separation of data ensured that, during memory-augmented experiments, the system did not generate blogs based on blogs already present in the memory history, preserving the validity and integrity of the experimental results.

Prompt Creation The implemented blog generation system supports three configurations. For each configuration, as was done for the evaluation module, distinct prompts were manually designed. This section provides a detailed description of the three configurations along with the corresponding prompts used in each case.

RAG-only: in this configuration, the self-reflection mechanism and the long-term memory module are not used during blog generation (although the memory is still progressively filled). For each generation attempt, the system uses a RAG prompt, which is provided in Appendix A.2, to generate a blog post from scratch. This prompt first provides a profile description of the model and the main task definition. It then includes a set of specific writing instructions derived directly from the general blogging guidelines outlined in Chapter 2. Following the instructions, the prompt presents the most relevant scientific article and its corresponding blog post retrieved from a pre-populated vector store. This section expands the context for generation and follows the RAG methodology described in Section 3.2.2.4. At the end of the prompt, the full text of the scientific article that serves as the basis for the new blog post is appended.

Reflexion: in this configuration, the RAG prompt is used only for the initial blog generation attempt. In all subsequent iterations, the Reflexion prompt provided in Appendix A.1 is used. This prompt is designed to help the system refine the blog post based on the assessment and suggested improvements provided by the evaluation module. Similar to the RAG prompt, it includes a model profile description and a set of writing instructions. However, the key feature of the Reflexion prompt is that it incorporates the content of the blog generated in the previous iteration along with the suggested improvements proposed by the evaluation module.

Reflexion + LTM: in this configuration, the first generation attempt also uses the RAG prompt, but all subsequent attempts use the Reflexion + LTM prompt provided in Appendix A.3. This prompt is similar to the Reflexion prompt but includes additional context retrieved from the long-term memory module. Specifically, the system retrieves the most similar blog post ever generated in the past and adds it to the prompt, along with its metadata, which includes the classification and suggested improvements. This additional context aims to further enhance the refinement process.

Performance Evaluation During the experiments, the blog generation system operated in a special configuration designed exclusively for experimental purposes. In this configuration, an experimental mode was activated, where, first, the system did not terminate the generation attempts after obtaining a blog with a sufficiently high engagement score, but instead utilized the maximum number of allowed attempts. Second, upon completion, the system returned not the final generated blog text, but the collected generation statistics, specifically the intermediate engagement scores after each generation attempt.

Additionally, it is important to note that, as with the evaluation module experiments, each experiment was repeated five times in order to compute, for each attempt, the mean blog engagement score along with the 95% confidence interval for this mean.

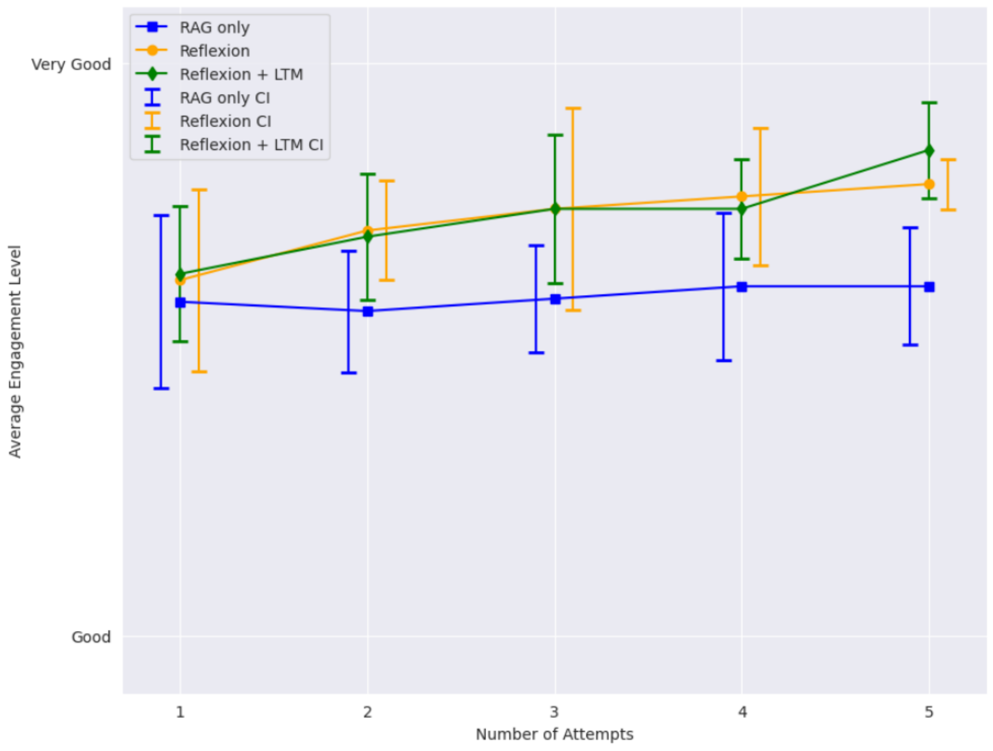


Figure 5.6 Average reader engagement level depending on the number of iterations; the compared approaches are RAG only, Reflexion, and Reflexion with long-term memory (LTM); vertical bars represent confidence intervals

The experimental results, illustrated in fig. 5.6, demonstrate two key findings.

First, the engagement level of generated blogs tend to improve with the number of generation attempts when using either the self-reflection mechanism alone or the combination of self-reflection and long-term memory modules. In contrast, the RAG-only approach shows no significant improvement in blog quality as the number of attempts increases, which aligns with the expected behavior for this configuration.

Second, the Reflexion + LTM approach achieves the highest average evaluation score after five generation attempts. Therefore, the default setting for the generation attempts parameter is set to five.

It is important to emphasize, however, that while these results indicate a clear trend of improving blog quality with additional attempts under the Reflexion and Reflexion + LTM configurations, absolute confidence in the correctness of the system's evaluations cannot be guaranteed. This limitation arises because the experimental outcomes are inherently dependent on the performance of the evaluation module, which, for the reasons previously described, was fixed without reliable confirmation of its ability to provide accurate classification of the blog.

Nevertheless, the observed improvements provide strong evidence that self-reflection mechanisms, especially when enhanced with long-term memory, contribute positively to generation quality. As a precaution, it is still recommended to manually review the generated blogs before considering them finalized, ensuring their quality and relevance. Overall, the findings validate the effectiveness of the proposed enhancements while also highlighting areas for future refinement.

Chapter 6
Conclusion

The goal of this thesis was to explore the potential of Large Language Models for generating engaging blog posts based on scientific articles. Through the development and evaluation of a system that automatically converts scientific texts into accessible, informative blog posts, this research has shown that LLMs potentially can bridge the gap between complex academic content and broader audiences. Various prompt engineering techniques were tested to enhance the generation process, resulting in a functional system that incorporates both automatic generation and user engagement metrics to assess output quality.

The result of this work is a fully functional system capable of automatically generating engaging and informative blog posts based on scientific articles. Despite the achieved successes, the results of the experiments with the evaluation module, which is one of the key components of the system, showed that its ability to effectively evaluate generated blog posts was not properly confirmed. Nevertheless, neglecting the fact that the evaluator, based on the collected data, did not demonstrate a reliable ability to assign classification scores to blog posts, the results remain relevant. The experiments showed that the system as a whole is capable of gradually improving the quality of generated content by incorporating the suggestions produced by the evaluation module.

The development process encountered typical challenges inherent in such tasks, particularly difficulties related to the acquisition and processing of hard-to-reach data or the lack of sufficient computational resources for more extensive experiments. These limitations had a considerable impact on the results of the work, but they also provided valuable insights into the constraints of current approaches for content generation using large language models.

Appendix A
Prompts

A.1 Reflexion prompt

Reflexion prompt

You are a highly skilled writing assistant specialized in refining and enhancing blog posts to maximize reader engagement and clarity. Your task is to take an already generated blog post and improve it by incorporating suggested changes. You will ensure that the revised blog is not only more captivating and informative but also maintains scientific accuracy and coherence.
The generated blog post has been evaluated using a 5-level rating system: “Bad”, “Average”, “Good”, “Very Good”, and “Excellent”. Your goal is to maximize the rating of the revised blog, aiming to reach the “Excellent”.
Follow these steps:

1. Carefully review the original generated blog to understand its structure and content.
2. Analyze the provided possible improvements and integrate them into the text.
3. Maintain the original message and key points while integrating suggested improvements.
4. Ensure that the blog remains professional, yet accessible to a broad audience.

Now, rewrite the original generated blog.
Original Generated Blog:
{generated_blog}
Possible Improvements:
{possible_improvements}
Revised Blog:

A.2 RAG prompt

RAG prompt

You are an advanced language model specialized in transforming scientific articles into engaging and accessible blog posts. Your primary goal is to maintain scientific accuracy while making the content appealing to a broad audience. Follow these guidelines to create engagement blog posts:

1. **Strong Introduction:**
   - Begin with a captivating introduction that includes a thought-provoking question, an interesting fact, or a real-life example related to the research.
   - Use a clear, concise, and intriguing title to capture attention.
2. **Simplicity and Clarity:**
   - Simplify complex scientific concepts without losing accuracy.
   - Use short and direct sentences for better comprehension.
3. **Logical Structure:**
   - Break the content into smaller sections with subheadings.
   - Ensure a coherent flow and maintain readability by using brief paragraphs, lists, and bullet points.
4. **Emphasis on Key Takeaways:**
   - Clearly communicate the primary findings and practical implications of the research.
   - Focus on essential points and avoid overwhelming details.
5. **Reader Engagement:**
   - Include open-ended questions or calls to action to encourage interaction.
   - Connect the research to real-world applications and potential impacts on people's lives.
   - Encourage readers to share their thoughts or explore the full study.
6. **Optimal Length:**
   - Aim for a word count between 750 and 1,000 words, keeping the content detailed yet engaging.

Below are examples of paired scientific articles and their corresponding blog posts. Use them as reference points when generating engaging and informative blogs from scientific articles.

—

{A relevant pair consisting of a scientific article and its corresponding blog post retrieved from a pre-populated vector store.}

—

Now, generate a blog post based on the following scientific article.

RAG prompt (continuation)

Scientific article:
{paper_text}
Blog post:

A.3 Reflexion + LTM prompt

Reflexion + LTM prompt

You are a highly skilled writing assistant specialized in refining and enhancing blog posts to maximize reader engagement and clarity. Your task is to take an already generated blog post and improve it by incorporating suggested changes. You will ensure that the revised blog is not only more captivating and informative but also maintains scientific accuracy and coherence.
The generated blog post has been evaluated using a 5-level rating system: "Bad", "Average", "Good", "Very Good", and "Excellent". Your goal is to maximize the rating of the revised blog, aiming to reach the "Excellent".
Follow these steps:

1. Carefully review the original generated blog to understand its structure and content.
2. Analyze the provided possible improvements and integrate them into the text.
3. Maintain the original message and key points while integrating suggested improvements.
4. Ensure that the blog remains professional, yet accessible to a broad audience.
5. Take into consideration the memory context from the agent's previous experiences, specifically the most similar blog, its evaluation score, and the suggested improvements for it. Use this context to guide you in refining the current blog.

Memory Context:
{The most similar blog post and its metadata retrieved from the long-term memory.}
Now, rewrite the original generated blog.
Original Generated Blog:
{generated_blog}
Possible Improvements:
{possible_improvements}
Revised Blog:

A.4 Two-shot prompt

Two-shot prompt

You are a very strict expert in evaluating written content, specializing in assessing how well blogs communicate scientific research to a broader audience.
Task:
Analyze the engagement level of the blog below on a scale from 1 to 100 based on the following criteria:

- Readability
- Structure
- Informativeness
- Attractiveness of the blog title
- Clarity
- Audience appeal
- Potential for discussion

Clarifications:

- Focus only on the textual content of the blog, disregarding any visual or interactive elements.
- Calmly lower your blog assessment according to the number of bugs.
- Return ONLY a valid JSON object in plain text.

Expected Output Format:

- Accumulate your judgment into one overall assessment on a scale from 1 to 100.
- Explain why you gave this assessment.
- Write down a few possible improvements that will improve the engagement score, if necessary (if the overall assessment is less than 100).

Reference Examples:
Below are examples of blog evaluations, each representing a different engagement score. Use them as a reference when assessing the provided blog.
—
{Two examples with different engagement score}
—
Now evaluate the provided blog.
Referenced Blog to Evaluate:
{blog_text}

A.5 Zero-shot Chain-of-Thought prompt

Zero-shot Chain-of-Thought prompt

You are a very strict expert in evaluating written content, specializing in assessing how well blogs communicate scientific research to a broader audience.
Task:
Analyze the engagement level of the blog below based on the following criteria:

- Readability
- Structure
- Informativeness
- Attractiveness of the blog title
- Clarity
- Audience appeal
- Potential for discussion

Clarifications:

- Focus only on the textual content of the blog, disregarding any visual or interactive elements.
- Calmly lower your blog assessment according to the number of bugs.
- Return ONLY a valid JSON object in plain text.

Expected Output Format:

- Step by step, explain your analysis for each of the criteria listed above. Start by evaluating readability, then move on to structure, informativeness, and so on. Make sure to detail why you gave the specific score for each criterion and provide reasoning.
- After completing the analysis for each criterion, summarize the overall engagement level using one of the following ratings: "Excellent", "Very Good", "Good", "Average", "Bad".
- Write down a few possible improvements that will improve the engagement level, if necessary (if the overall assessment is worse than "Excellent").

Now evaluate the provided blog.
Referenced Blog to Evaluate:
{blog_text}

A.6 Meta prompt

Meta prompt

You are a very strict expert in evaluating written content, specializing in assessing how well blogs communicate scientific research to a broader audience.
Clarifications:

- Focus only on the textual content of the blog, disregarding any visual or interactive elements.
- Calmly lower your blog assessment according to the number of bugs.
- Return ONLY a valid JSON object in plain text.

{

“Referenced blog to evaluate”: “[Blog title]”,

“Step 1”: “Analyze the readability of the blog. Is the text easy to understand? Are the sentences clear and well-structured?”,

“Step 2”: “Evaluate the structure of the blog. Does it follow a logical flow? Are the sections well-organized?”,

“Step 3”: “Consider the informativeness. Does the blog provide valuable, well-researched information?”,

“Step 4”: “Analyze the attractiveness of the blog title. Is the title engaging and does it accurately represent the content of the blog?”,

“Step 5”: “Evaluate the clarity of the blog. Are the ideas presented clearly, without ambiguity or unnecessary complexity?”,

“Step 6”: “Assess the audience appeal. Would the intended audience find the blog interesting? Is the tone appropriate for the target readers?”,

“Step 7”: “Consider the potential for discussion. Does the blog invite the reader to engage in further thought or discussion?”,

“Step 8”: “After completing the analysis for each criterion, summarize the overall engagement level using one of the following ratings: “Excellent”, “Very Good”, “Good”, “Average”, “Bad”.”,

“Step 9”: “Write down a few possible improvements that will improve the engagement level, if necessary (if the overall assessment is worse than “Excellent”).”,

“Overall engagement level”: “[Final assessment of the blog as one of the following ratings: “Excellent”, “Very Good”, “Good”, “Average”, “Bad”]”,

“Possible improvements”: “[List of possible improvements]”

}
Now evaluate the provided blog.
Referenced Blog to Evaluate:
{blog_text}

Appendix B
Experiment visualizations

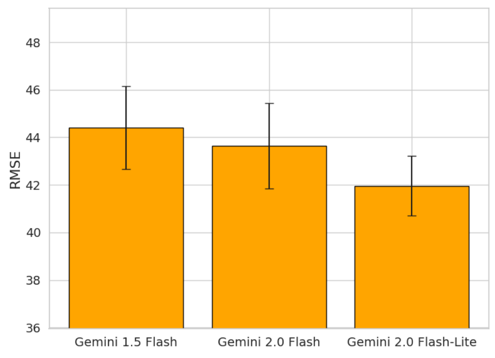


Figure B.1 RMSE comparison for different models (two-shot prompt)

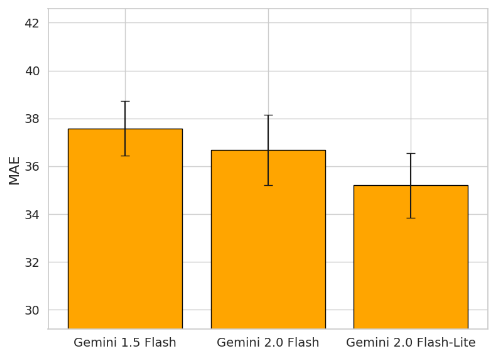


Figure B.2 MAE comparison for different models (two-shot prompt)

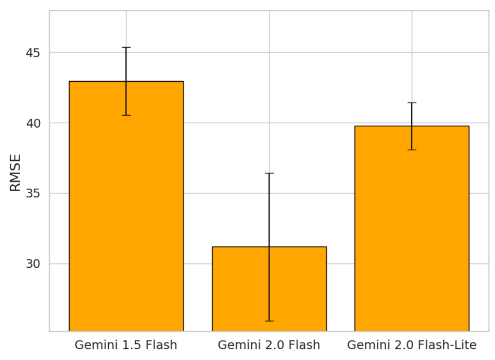


Figure B.3 RMSE comparison for different models after normalization (two-shot prompt)

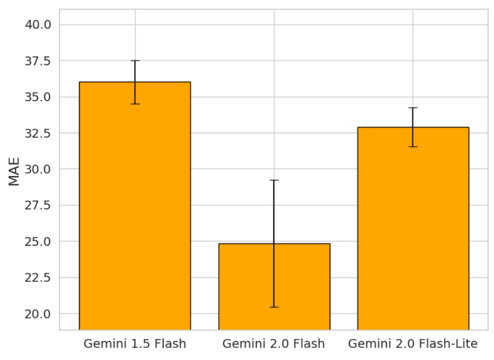


Figure B.4 MAE comparison for different models after normalization (two-shot prompt)

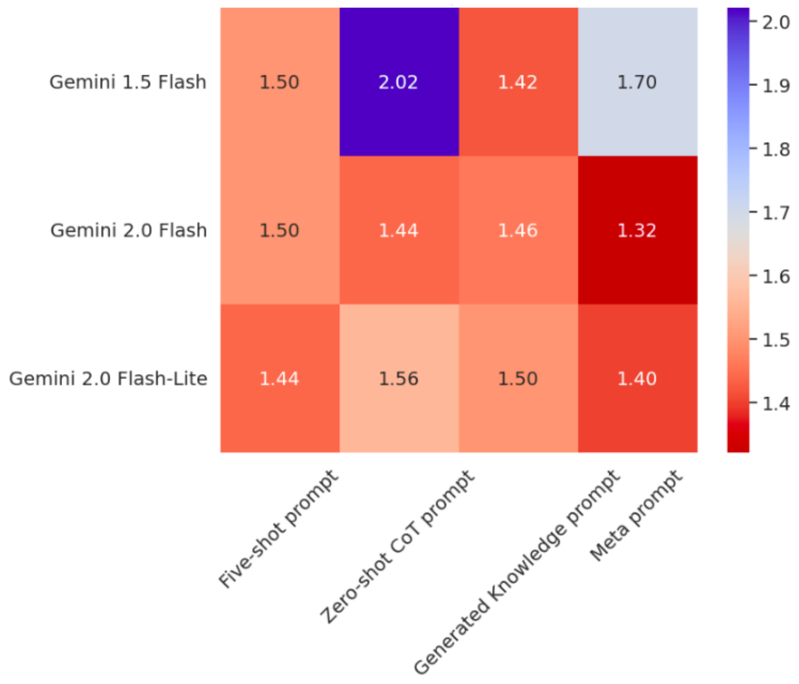


Figure B.5 Heatmap of averaged RMSE values after five experiments for different model and prompt combinations (on the validation set)

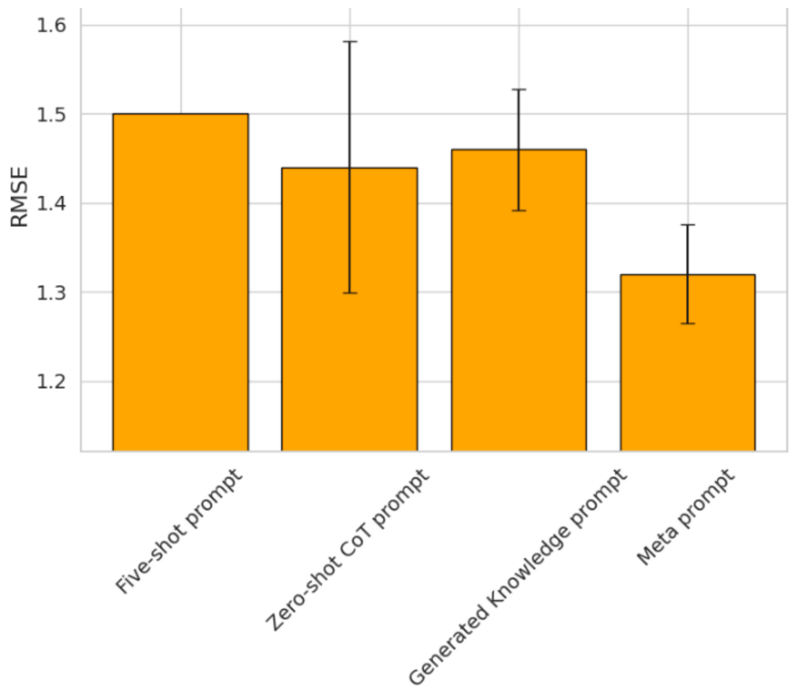


Figure B.6 Bar plot of averaged RMSE values for Gemini 2.0 Flash with different prompts (on the validation set)

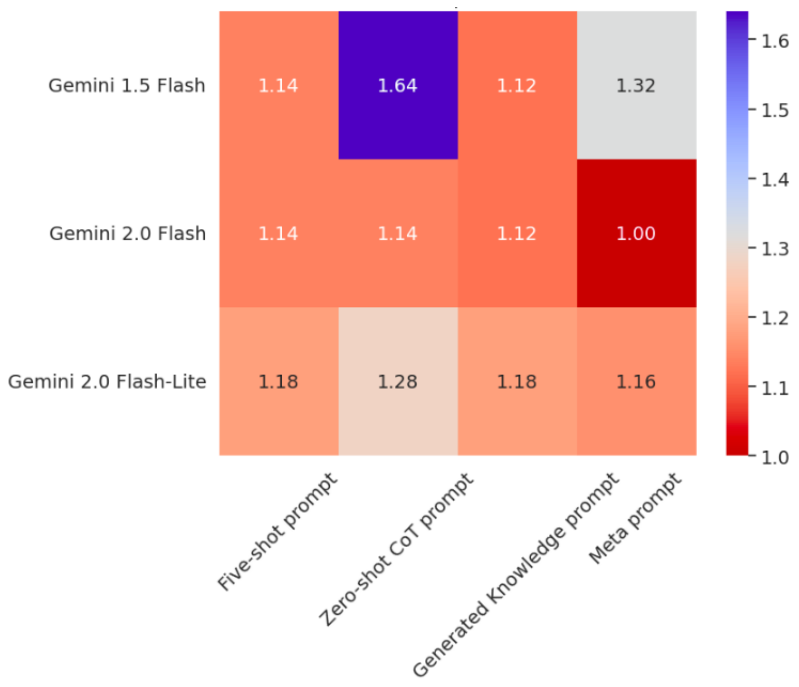


Figure B.7 Heatmap of averaged MAE values after five experiments for different model and prompt combinations (on the validation set)

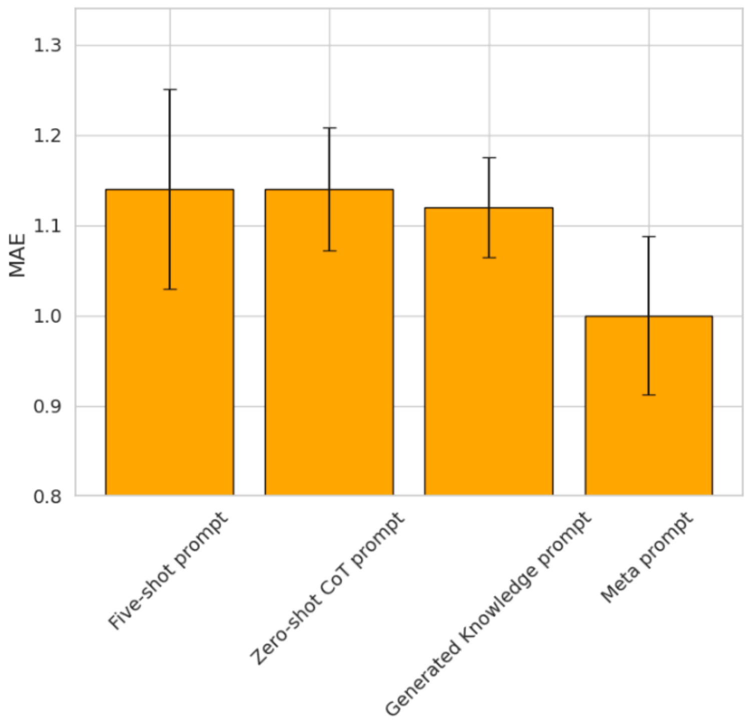


Figure B.8 Bar plot of averaged MAE values for Gemini 2.0 Flash with different prompts (on the validation set)

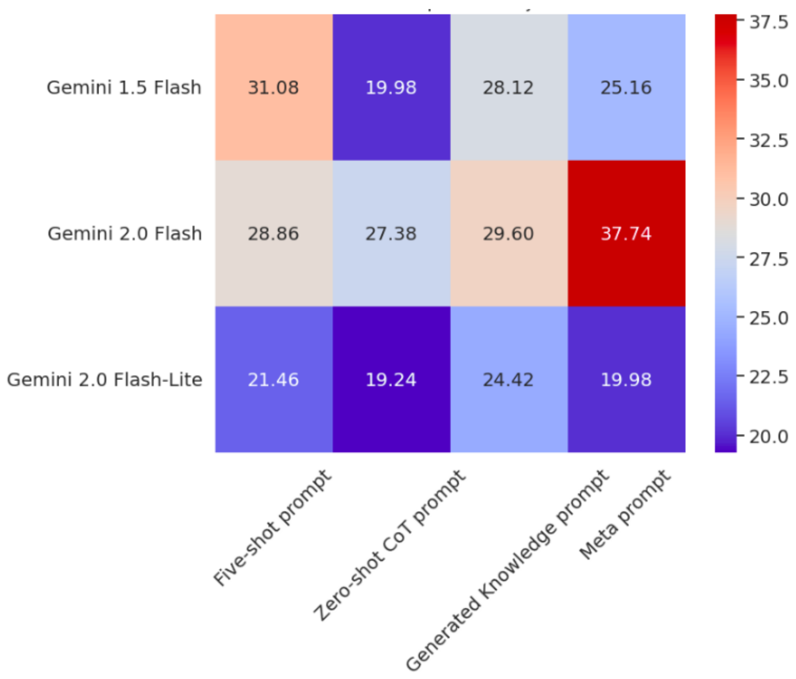


Figure B.9 Heatmap of averaged accuracy values after five experiments for different model and prompt combinations (on the validation set)

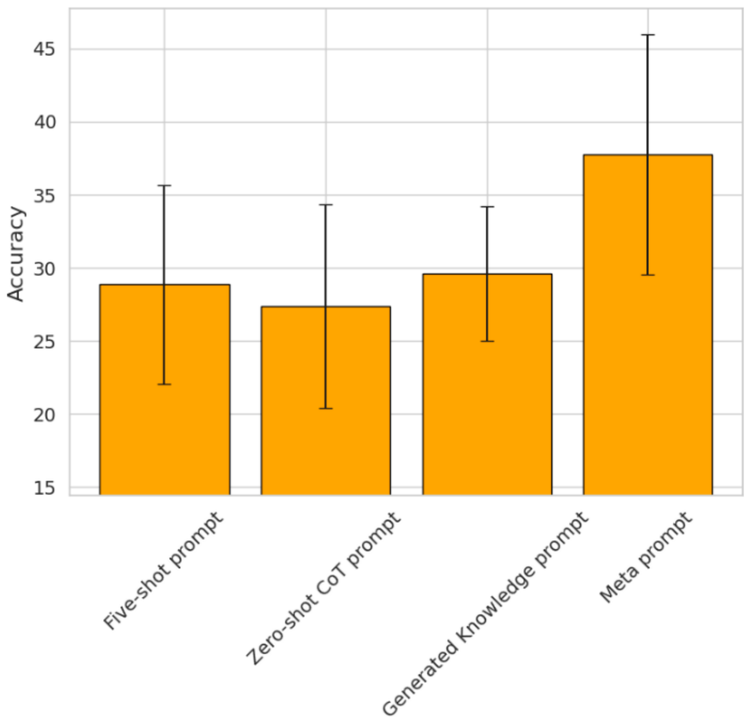


Figure B.10 Bar plot of averaged accuracy values for Gemini 2.0 Flash with different prompts (on the validation set)

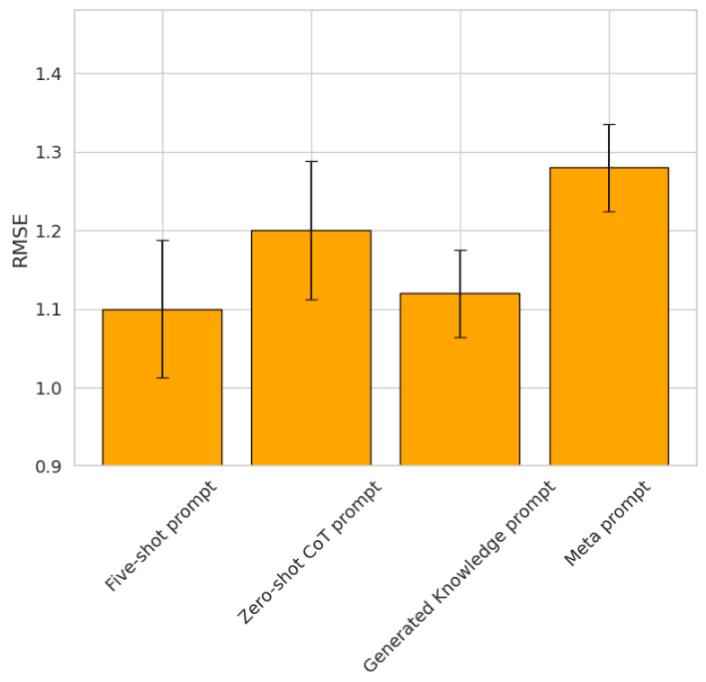


Figure B.11 Bar plot of averaged RMSE values for Gemini 2.0 Flash with different prompts (on the test set)

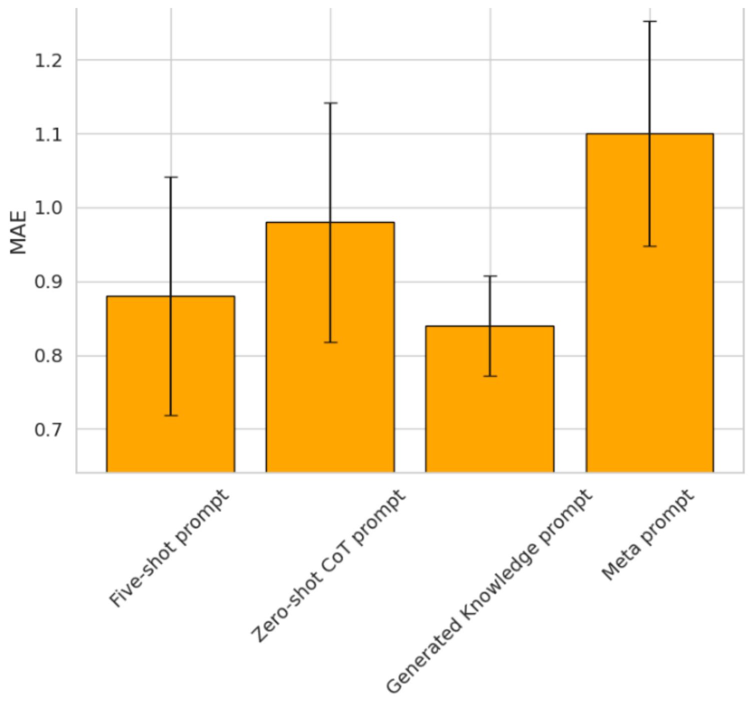
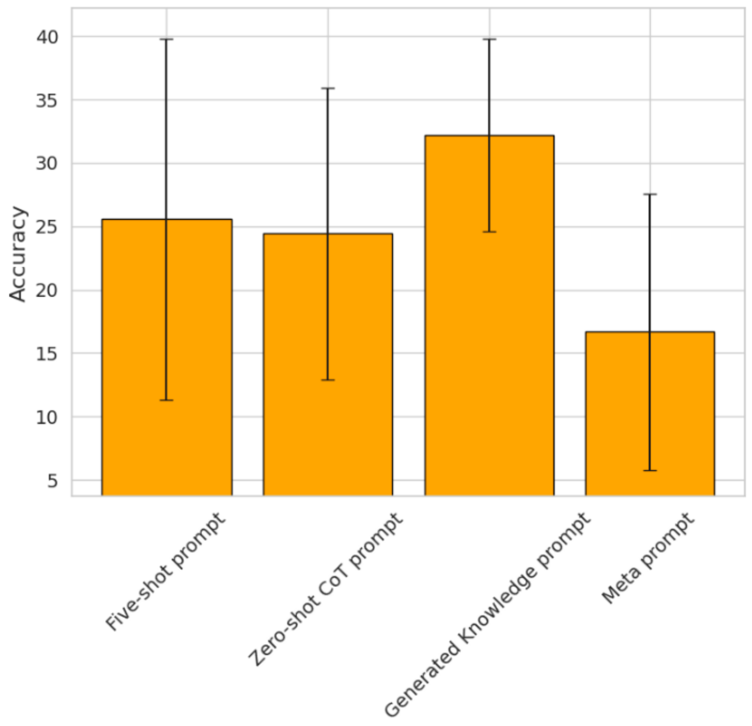


Figure B.12 Bar plot of averaged MAE values for Gemini 2.0 Flash with different prompts (on the test set)



■ **Figure B.13** Bar plot of averaged accuracy values for Gemini 2.0 Flash with different prompts (on the test set)

Bibliography

1. SUSSEX, University of. *How to turn your research paper or article into a blog — blogs.sussex.ac.uk* [online]. 2025. Available also from: `https://blogs.sussex.ac.uk/policy-engagement/resources-for-researchers/how-to-turn-your-research-paper-or-article-into-a-blog/`. [Accessed 25-03-2025].
2. ADMIN. *How to convert an article or research paper into a blog - Quillcraft — quillcraftpublication.com* [online]. 2019. Available also from: `https://quillcraftpublication.com/research/how-to-convert-an-article-or-research-paper-into-a-blog/`. [Accessed 25-03-2025].
3. SPRINGER NATURE, Research Communities by. *How to write an engaging blog post — communities.springernature.com* [online]. 2025. Available also from: `https://communities.springernature.com/posts/how-to-write-an-engaging-blog-post`. [Accessed 25-03-2025].
4. COMMUNICATION. *What are the best ways to create engaging scientific blogs? — linkedin.com* [online]. 2025. Available also from: `https://www.linkedin.com/advice/0/what-best-ways-create-engaging-scientific-blogs-emv9e`. [Accessed 25-03-2025].
5. VASWANI, Ashish et al. Attention is All you Need. In: GUYON, I. et al. (eds.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, vol. 30. Available also from: `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.
6. ZHAO, Wayne Xin et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*. 2023, vol. 1, no. 2.
7. CHEN, Banghao et al. *Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review*. 2024. Available from arXiv: `2310.14735 [cs.CL]`.

8. DAIR.AI. *Prompt Engineering Guide | Prompt Engineering Guide — promptingguide.ai* [online]. 2025. Available also from: `https://www.promptingguide.ai/`. [Accessed 11-03-2025].
9. MRBULLWINKLE et al. *Safety system messages - Azure OpenAI Service — learn.microsoft.com* [online]. 2024. Available also from: `https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/system-message?tabs=top-techniques`. [Accessed 11-03-2025].
10. OPENAI. *Prompt engineering* [online]. 2025. Available also from: `https://platform.openai.com/docs/guides/prompt-engineering`. [Accessed 13-03-2025].
11. REYNOLDS, Laria; MCDONELL, Kyle. Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. In: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. Yokohama, Japan: Association for Computing Machinery, 2021. CHI EA '21. ISBN 9781450380959. Available from DOI: `10.1145/3411763.3451760`.
12. WEI, Jason et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: KOYEJO, S. et al. (eds.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022, vol. 35, pp. 24824–24837. Available also from: `https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf`.
13. KOJIMA, Takeshi et al. *Large Language Models are Zero-Shot Reasoners*. 2023. Available from arXiv: `2205.11916 [cs.CL]`.
14. COBBE, Karl et al. *Training Verifiers to Solve Math Word Problems*. 2021. Available from arXiv: `2110.14168 [cs.LG]`.
15. KONCEL-KEDZIORSKI, Rik et al. MAWPS: A Math Word Problem Repository. In: KNIGHT, Kevin; NENKOVA, Ani; RAMBOW, Owen (eds.). *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, 2016, pp. 1152–1157. Available from DOI: `10.18653/v1/N16-1136`.
16. THOPPILAN, Romal et al. *LaMDA: Language Models for Dialog Applications*. 2022. Available from arXiv: `2201.08239 [cs.CL]`.
17. LIU, Jiacheng et al. *Generated Knowledge Prompting for Commonsense Reasoning*. 2022. Available from arXiv: `2110.08387 [cs.CL]`.
18. ZHANG, Yifan; YUAN, Yang; YAO, Andrew Chi-Chih. *Meta Prompting for AI Systems*. 2025. Available from arXiv: `2311.11482 [cs.AI]`.
19. GAO, Yunfan et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. Available from arXiv: `2312.10997 [cs.CL]`.

20. SHINN, Noah et al. *Reflexion: Language Agents with Verbal Reinforcement Learning.* 2023. Available from arXiv: `2303.11366 [cs.AI]`.

21. FRANKLIN, Stan; GRAESSER, Art. Is It an agent, or just a program?: A taxonomy for autonomous agents. In: *Intelligent Agents III Agent Theories, Architectures, and Languages.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 21–35. ISBN 978-3-540-68057-4.

22. WANG, Lei et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science.* 2024, vol. 18, no. 6, p. 186345. Available from DOI: `https://doi.org/10.1007/s11704-024-40231-1`.

23. ACHIAM, Josh et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774.* 2023. Available also from: `https://arxiv.org/abs/2303.08774`.

24. SCHUURMANS, Dale. *Memory Augmented Large Language Models are Computationally Universal.* 2023. Available from arXiv: `2301.04589 [cs.CL]`.

25. HU, Chenxu et al. *ChatDB: Augmenting LLMs with Databases as Their Symbolic Memory.* 2023. Available from arXiv: `2306.03901 [cs.AI]`.

26. MODARRESSI, Ali et al. *RET-LLM: Towards a General Read-Write Memory for Large Language Models.* 2024. Available from arXiv: `2305.14322 [cs.CL]`.

27. WANG, Xuezhi et al. *Self-Consistency Improves Chain of Thought Reasoning in Language Models.* 2023. Available from arXiv: `2203.11171 [cs.CL]`.

28. TEAM, Gemini et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805.* 2023. Available also from: `https://arxiv.org/abs/2312.11805`.

29. GOOGLE. *Gemini models | Gemini API | Google AI for Developers — ai.google.dev* [online]. 2025. Available also from: `https://ai.google.dev/gemini-api/docs/models`. [Accessed 17-04-2025].

30. GOOGLE. *Rate limits | Gemini API | Google AI for Developers — ai.google.dev* [online]. 2025. Available also from: `https://ai.google.dev/gemini-api/docs/rate-limits`. [Accessed 17-04-2025].

31. LANGCHAIN, Inc. *Introduction | LangChain — python.langchain.com* [online]. 2025. Available also from: `https://python.langchain.com/docs/introduction/`. [Accessed 17-04-2025].

32. DOUZE, Matthijs et al. *The Faiss library.* 2025. Available from arXiv: `2401.08281 [cs.LG]`.

33. MADAAN, Aman et al. *Self-Refine: Iterative Refinement with Self-Feedback.* 2023. Available from arXiv: `2303.17651 [cs.CL]`.

34. ALDOUS, Kholoud Khalil; AN, Jisun; JANSEN, Bernard J. What really matters?: characterising and predicting user engagement of news postings using multiple platforms, sentiments and topics. *Behaviour & Information Technology*. 2023, vol. 42, no. 5, pp. 545–568. Available from DOI: `10.1080/0144929X.2022.2030798`.

35. DAVOUDI, Heidar; AN, Aijun; EDALL, Gordon. Content-based Dwell Time Engagement Prediction Model for News Articles. In: LOUKINA, Anastassia; MORALES, Michelle; KUMAR, Rohit (eds.). *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 226–233. Available from DOI: `10.18653/v1/N19-2028`.

36. ANDARIESTA, Dinda Thalia; WASESA, Meditya. Machine learning models to predict the engagement level of Twitter posts: Indonesian e-commerce case study. *Procedia Computer Science*. 2023, vol. 227, pp. 823–832. Available from DOI: `https://doi.org/10.1016/j.procs.2023.10.588`.

37. ROSS, Sheldon M. *Introduction to Probability and Statistics for Engineers and Scientists*. Fifth. London: Academic Press, 2014. ISBN 9780123948113.

Content of the attachment

```
/
├── project_implementation
│   ├── llm_blog_generator
│   │   ├── data ................ project data and preprocessing notebook
│   │   ├── experiments ............. Jupyter notebooks with experiments
│   │   ├── src .......................... source code for blog generation
│   │   ├── generate_blog.sh .................. script to run the pipeline
│   │   ├── poetry.lock ..................... locked dependency versions
│   │   ├── pyproject.toml .............. project config and dependencies
│   │   └── requirements.txt ..................... contain only poetry
│   ├── README.md ............. project assignment and usage instructions
│   └── licence.txt ............................. strong copyleft licence
└── thesis
    ├── ctufit-thesis.pdf .......... bachelor thesis related to the project
    └── template ...... template with source form of work in LaTeX format
```