CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Automatic Engineering Report Generation Using Multi-Agent Systems and Large Language Models

Master's thesis in Computer science and engineering

Alexander Nihlstrand
Gustav Lundberg

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Automatic Engineering Report Generation Using Multi-Agent Systems and Large Language Models

Alexander Nihlstrand
Gustav Lundberg

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Automatic Engineering Report Generation Using Multi-Agent Systems and Large Language Models
Alexander Nihlstrand
Gustav Lundberg

Supervisor: Moa Johansson, Computer Science and Engineering
Advisor: Elias Sörstrand, Volvo Trucks
Examiner: Marina Axelsson-Fisk, Mathematical Sciences

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Automatic Engineering Report Generation Using Multi-Agent Systems and Large Language Models
Alexander Nihlstrand
Gustav Lundberg
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigates the development and evaluation of a Multi-Agent Systems designed to automatically generate technical reports from unstructured truck testing data, using LLMs and a RAG based framework. The study explores how various architectural configurations and prompting strategies, such as Chain-of-Thought reasoning, one-shot prompting, and partitioned RAG artifacts, affect the factual accuracy, structural quality, and practical utility of the generated reports in an engineering context.

Performance is assessed through a comparative analysis of three reports: a Multi-Agent System generated, single-agent baseline, and human-written report. Evaluations involve both human evaluators and LLM-based scoring, supplemented by a RAGAS pipeline for measuring content reliability and relevance. Human evaluations indicate that the Multi-Agent System performs well in terms of structural coherence and linguistic fluency but lacks the analytical depth and data interpretation accuracy of human-authored reports.

Despite being outperformed by human-written reports across most human evaluation metrics, the Multi-Agent System demonstrates promising capabilities. It generates fluent, well-organized text and can identify and categorize key technical events. However, the Multi-Agent System also exhibits significant limitations, particularly in domain-specific reasoning and deeper factual understanding, which are critical components of technical report writing.

Finally, the study reveals key challenges in evaluation itself. NLP tasks are inherently subjective and difficult to evaluate due to the absence of a ground truth. Furthermore, discrepancies between human and LLM-based assessments suggest that each favors different qualities: while LLMs emphasize coherence and alignment with retrieved artifacts, human evaluators prioritize domain relevance, contextual understanding, and deeper analysis. This divergence underscores the subjective and multifaceted nature of language evaluation, especially within highly technical domains.

Keywords: LLMs, Agents, Multi-Agent System, AI, Generative AI, Prompting Techniques, RAG, NLP Evaluation, Report Generation.

Acknowledgements

We would like to express our gratitude to our academic supervisor, Moa Johansson from Chalmers University of Technology, for her invaluable feedback and support throughout the project. Her assistance in report writing, mathematical insights, and overall guidance has been highly important for the quality of our work.

We also extend a thank you to our advisor at Volvo Technology, Elias Sörstrand, for his expertise in generative AI and agent-based systems, helpful brainstorming sessions, and for providing access to essential resources and extensive domain knowledge.

Finally, we would also like to thank our manager, Behrooz Razaznejad, for making this thesis possible, as well as the Volvo experts who took the time to participate in the evaluation of our system. Your input has been crucial for the evaluation of the system and thereby this thesis.

Alexander Nihlstrand & Gustav Lundberg, Gothenburg, 2025-06-11

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence. 1, 2, 5

ANN Approximate Nearest Neighbor. 9

CoT Chain-of-Thought. v, 11, 31, 47

DoT Degeneration-of-Thought. 34

ER Engineering Report. 1–3

IP Inner Product. 18, 23

LLM Large Language Model. xiii, 1–3, 5, 6, 8–12, 14, 15, 17–20, 23, 25, 33, 41, 47, 49, 53–55, 57

MADS Multi-Agent Debate and Self-Consistency. 34

MAS Multi-Agent System. v, 1–3, 12, 15, 19, 21, 22, 27, 29, 41, 42, 45, 49–52, 54, 55, 57

MoE Mixture-of-Experts. 7, 8

NLG Natural language generation. 3, 13, 31

NLP Natural Language Processing. 5, 8

QA Question Answering. 13

RAG Retrieval-Augmented Generation. xiii, 9, 14, 20, 21, 31, 47, 56

RAGAS Retrieval-Augmented Generation Assessment Suite. 13, 14, 31, 39, 41, 47, 49

1

Introduction

This thesis is conducted in collaboration between Chalmers University of Technology and Volvo Technology, and investigates how Large Language Model (LLM)-based Multi-Agent Systems (MASs) can be used to automate the generation of engineering reports produced in connection with truck testing activities. At Volvo's Powertrain Strategic Development, engineers carry out extensive truck tests to evaluate new powertrain technologies, ensuring compliance with performance, efficiency, and regulatory standards. However, the process of documenting test results in comprehensive Engineering Reports (ERs) remains largely manual and time-consuming. It requires engineers to analyze large volumes of both structured and unstructured data, synthesize their findings, and compile reports that serve as a foundation for critical technological and business decisions.

While generative Artificial Intelligence (AI), particularly LLMs, shows strong potential in processing technical content and generating coherent text, their application in domain-specific, high-stakes industrial settings remains limited. Most current approaches use monolithic LLMs that lack role specialization. This limits their ability to manage technically complex tasks and generate reliable, multi-faceted documents such as ERs. A key challenge, therefore, is to develop system architectures that reflect the distributed expertise and task decomposition of human engineering teams. To address this, the thesis proposes a multi-agent framework in which specialized LLMs are assigned to distinct sub-tasks or report sections and collaborate through structured communication and coordination.

To this end, the thesis aims to automate the report writing process to reduce the workload on engineers, enabling them to focus on higher-value tasks. The proposed methodology not only supports Volvo Technology's electrification efforts but also holds relevance for other industries that rely on extensive technical documentation or data-driven insights. Moreover, this work advances the understanding of generative AI, LLM optimization, and MAS coordination, with a strong emphasis on applied implementation. It addresses a key gap in the literature by presenting a practical, domain-specific integration of LLMs within a multi-agent framework for ER generation.

1.1 Background

Recent studies have demonstrated the effectiveness of generative AI, particularly LLMs, in automating report generation across diverse domains[1, 2]. In the healthcare sector, Park et al. [3] employed GPT-3.5-turbo in a zero-shot-setting to produce MRI diagnostic reports, achieving impressive accuracy and minimal factual inconsistencies through structured prompting and rigorous human validation. Similarly, in the public sector, Gupta et al. [4] integrated specialized LLMs into semi-automated policy report generation systems, facilitating efficient data processing, summarization, and visualization. While these systems demonstrate strong performance in semi-autonomous settings with well-defined task boundaries, their applicability to more complex, technical domains, such as engineering, remains limited.

In light of these constraints, recent research has further questioned the viability of LLMs in more demanding domains such as ER-generation. Pinto et al. [5] showed that while generative AI models such as GPT-3.5 can produce outputs with promising clarity and coherence, they generally do not surpass human authors in terms of nuanced writing and interpretive depth. In response, MASs have emerged as a proposed solution. These systems hold the theoretical promise of outperforming single-agent models on complex tasks that require specialization and coordination [6]. Yet, early empirical studies suggest that MASs often yield only marginal improvements over single-agent models, particularly on standard benchmarks, and may instead introduce new points of failure as system complexity grows. As such, several challenges must be addressed before these systems can be reliably deployed in domain-specific applications.

A central challenge in MAS architectures concerns inter-agent alignment and coordination. Similar to human teams, agent-based systems require clearly defined roles and communication protocols to work effectively. Prior research has identified a range of failure modes in MAS implementations, including unclear objectives, conflicting behaviors, and poorly managed verification steps [6]. Of particular concern is the problem of conversation memory loss, when agents fail to preserve or update a shared understanding of the task context, leading to inconsistent or contradictory outputs. Although techniques such as shared memory states and workflow planning have been proposed to address these issues, maintaining coherent coordination over long task sequences remains a non-trivial challenge.

These coordination challenges are further compounded by the risk of error propagation in MAS architectures. LLMs are known to occasionally generate inaccurate or misleading content, a phenomenon known as hallucination, which is well documented in single-agent settings [7]. However, in a multi-agent setup, a hallucinated detail introduced by one agent may still be treated as factual by others, thereby amplifying the error as information flows between agents [8]. This is further complicated by the fact that LLMs typically lack calibrated uncertainty, where they do not naturally signal when they are uncertain about their outputs [9]. As a result, agents may confidently propagate false information which can become particularly problematic in applications such as engineering report generation, where factual accuracy is critical.

A final challenge lies in evaluating outputs from LLM-based systems. Since Natural language generation (NLG) lacks a clear ground truth, particularly in tasks such as report writing where multiple outputs may be equally valid, objective assessment becomes inherently difficult. Common automatic metrics, such as BLEU and ROUGE, correlate poorly with human judgment [10]. Although recent LLM-based evaluators such as G-EVAL offer promising, reference-free alternatives, they remain sensitive to prompt design and may exhibit bias toward LLM-generated content [11]. These limitations underscore the need for applied studies that not only address generation and coordination, but also propose practical strategies for evaluating system performance in real-world settings.

1.2 Research Questions

In the light of these challenges, this thesis aims to explore whether a specialized MAS can improve the efficiency and quality of automated ER generation. This leads to the following research questions:

> **RQ1:** *Can a specialized MAS, with clearly defined agent roles and coordination mechanisms, improve the quality and efficiency of engineering report generation, as measured by expert evaluations and automated text metrics such as fluency, factuality, repetition, structure, and usefulness, compared to hand-written reports?*
>
> **RQ2:** *To what extent can prompting techniques reduce hallucinations and improve the performance of an LLM-based MAS, as measured by automatic evaluation metrics, such as faithfulness, factual correctness, and summarization score, provided through the RAGAS framework?*
>
> **RQ3:** *How does automatic LLM-based evaluation compare to human evaluation in assessing the quality of a generated ER, particularly within the context of NLG tasks, with a focus on consistency, comparability, and correlation between evaluation outcomes?*

2

Theory

The recent advancement of LLMs has paved the way for new AI-driven applications, which have consequently improved in performance. The following sections provides an overview of LLMs, the emergence of LLM-based agents, as well as techniques aimed at enhancing the quality and adaptability of agent based workflows. Lastly, it also presents a number of potential evaluation frameworks, utilized in Natural Language Processing (NLP) tasks.

2.1 Large Language Models

LLMs are artificial intelligence systems designed to understand, generate, and manipulate human language. These models are built upon the *transformer* architecture introduced by Vaswani et al. [12] and utilizes an attention mechanism, which enables LLMs to selectively focus on relevant parts of an input sequence when generating or interpreting text. Unlike earlier recurrent architectures, transformers allow for significantly improved parallelization, which makes them especially effective for language tasks.

2.1.1 LLM Training

LLMs are trained on large-scale corpora comprising text from diverse sources such as books, websites, and other text-based content, to capture statistical patterns, word relationhips, and semantic structures. This pretraining phase is typically framed as a language modeling task, where the model learns to predict the next token in a sequence given the previous tokens [13]. This is described as an autoregressive modeling objective, formally described as:

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1})$$

The training process minimizes the cross-entropy loss between the predicted and actual next tokens according to:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t | x_1, \ldots, x_{t-1})$$

where T is the sequence length and $P(y_t|x_1, \ldots, x_{t-1})$ represents the model's predicted probability distribution over the vocabulary at step t [14].

Importantly, LLMs are probabilistic models: they do not produce deterministic outputs but rather sample from a probability distribution over possible continuations. As such, multiple outputs may be valid for the same input prompt. Furthermore, this probabilistic nature means that outputs are not guaranteed to be factual or consistent. In particular, LLMs have a recurring issue with hallucinations, meaning confidently producing text that appears fluent but is factually incorrect or fabricated. A key parameter for controlling this variability is the *temperature*, which adjusts the entropy of the probability distribution during generation. Lower temperatures (e.g., close to 0) make the model more deterministic, while higher temperatures (e.g., 0.8 or above) increase randomness and creativity in the output [15].

2.1.2 In-context learning

After pretraining, LLMs can be adapted to specific tasks or domains through *fine-tuning*, which involves continued training on a more targeted dataset. Fine-tuning updates the model parameters to specialize in a narrower function, such as legal reasoning, medical analysis, or dialogue generation. This phase can be supervised (using labeled datasets) or unsupervised (with domain-specific corpora), and has been shown to significantly improve performance on downstream applications [16, 17].

Alternatively, models can be adapted without changing parameters through *in-context learning*, where task instructions or examples are included directly in the prompt at inference time. This approach leverages the model's generalization capabilities without additional training [18], making it especially suitable for scenarios where retraining is computationally infeasible or model weights are inaccessible. Given the scope of this thesis and the practical constraints on high quality data, an in-context learning approach is employed instead of fine-tuning.

2.2 LLM specifications

The recent advancement of LLMs has produced a diverse ecosystem of model families, each optimized for different applications and requirements. Although most LLMs are originally based on the *transformer* architecture, substantial variation has emerged in areas such as parameter scale, training methodology, and inference strategies. These differences reflect competing priorities in the design of LLMs, which directly affects areas such as performance, efficiency, and accessibility. The following section provide an overview of LLM design dimensions.

2.2.1 LLM Design Dimensions

A fundamental design factor for LLMs is the parameter scale. The number of trainable parameters determines a model's capacity to learn complex patterns and generalize from data [19]. Larger models typically exhibit improved performance

on tasks requiring abstract reasoning, contextual understanding, and multi-step inference, as the increased parameter count enables richer internal representations and a greater capacity for capturing linear and nonlinear dependencies. However, larger models come with higher demands on memory and computational resources, which increases both operational cost and inference time.

To mitigate high computational costs, a focus on efficiency has emerged. Techniques such as Mixture-of-Experts (MoE) architectures [20, 21] and sparsity-aware attention mechanisms [22] enable models to reduce the number of active parameters during inference, which lowers computational cost while preserving performance. MoE models, for instance, activate only a subset of expert layers for each token, allowing for substantial computational savings without sacrificing the total parameter count. Sparsity-aware attention mechanisms instead dynamically adjust the attention pattern, selectively processing only relevant parts of the input sequence.

Finally, *accessibility* plays a critical role in how LLMs are deployed and used. Open-source models, such as LLaMA [23] and DeepSeek [24], provide public access to model weights, enabling easier reproducibility, fine-tuning and transparency in usage. In contrast, proprietary models like GPT-4 [25] are only accessible via API endpoints, limiting customization and deeper understanding of the models.

2.2.2 LLM Specifications

For this thesis four LLMs are utilized, with background in their different specifications and design dimensions. A detailed description of these models is provided below, accompanied by an overview summarized in Table 2.1.

`Phi-4` is a 14 billion parameter Transformer-based model introduced by Microsoft Research [26]. It is trained on a mixture of high-quality synthetic and filtered organic datasets, with a focus on enhancing reasoning and problem-solving capabilities, particularly for STEM-related tasks. Phi-4 is optimized for generating coherent and contextually rich responses with extended context length support, making it suitable for complex and structured problem-solving applications. The model operates on text-only modality and is publicly available with open access to its weights and configuration files.

`LLaMA-33-70b` is part of the family of the transformer models released by Meta AI, incorporating 70B parameters. This model is pretrained on a dataset consisting of publicly available and licensed text and is further fine-tuned for instruction-following [23]. LLaMA models are optimized for efficient, large-scale inference, focusing on delivering high-quality text generation for a variety of tasks. Designed primarily for text-only processing, LLaMA models are accessible for both research and commercial use.

`GPT-4`, developed by OpenAI, is a multimodal model capable of processing both text and image inputs. GPT-4 is trained using a combination of publicly available and proprietary datasets, showcasing high performance across a broad range of standardized language tasks. It is particularly strong in generating detailed, context-rich responses and demonstrates advanced capabilities in tasks requiring nuanced

understanding and complex reasoning. GPT-4 is distributed via API access only, with model weights and training data not publicly released [25].

`DeepSeek-R1-70B` is a dense transformer model based on the LLaMA 3.3 architecture, comprising 70 billion parameters, which are all active during inference. It serves as a distilled version of the larger DeepSeek-R1 MoE model, with strengths in code generation, mathematics, and structured reasoning. It also pertains the large context window of the larger R1 model, with an ability to consider 128 000 token inputs [27]. Unlike the DeepSeek-R1, the 70B variant uses a standard dense architecture, offering strong performance with lower complexity. It is a text-only model with publicly available weights and training configurations, making it accessible for research.

Table 2.1: Overview of Evaluated Models Including Parameter Size, Supported Modalities, and Key Architectural Features

Model	**Params**	**Modality**	**Key Features**
Phi-4	14B	Text	Efficient, structured reasoning.
LLaMA-33-70b	70B	Text	Large, instruction-tuned.
GPT-4	Proprietary	Text & Image	Multimodal, advanced reasoning.
DeepSeek-R1-70b	70b	Text	Efficient, large context window.

2.3 Word Embeddings

Word embeddings are a foundational component in modern NLP, enabling the representation of words, phrases, or entire documents as dense vectors in a continuous, high-dimensional space. These vector representations capture semantic relationships and contextual meaning within the data, making them ideal for storing sequences of textual information. The following section will further explore word embeddings and their applications in an LLM context.

2.3.1 Embedding Models

Unlike traditional sparse representations such as one-hot encoding, word embeddings allow models to understand linguistic relationships based on proximity in a vector space. This approach allows for the encoding of semantic and syntactic information, where the distance and direction between vectors reflect the relationships between words. Two common embedding methods are Word2Vec [28] and GloVe [29], which generate embeddings that preserve semantic similarity, so that words or phrases with similar meanings are located close to each other in vector space. More advanced contextual embedding models like BERT and Sentence-BERT [30] go further by accounting for word meaning in context, dynamically adjusting embeddings depending on surrounding text. As a result, embeddings for related words like “engine” and “motor” are located closer to each other than to unrelated words like “banana”, reflecting their similar meanings in a given domain. The embedding model utilized in this thesis is BGE-M3 [31], which is capable of tasks such as semantic search and multilingual retrieval, both for short and long input texts.

2.3.2 Vector Databases

Vector databases such as FAISS [32], Pinecone [33], and Milvus [34] are optimized for storing and querying large collections of high-dimensional vectors. In a truck testing context, these embeddings can represent key information, such as error descriptions, solutions, summaries, or even entire logbook entries. Each data-point is embedded using a suitable embedding model, and the resulting vectors are indexed within the vector database for fast retrieval. The retrieval process typically uses distance metrics such as *cosine similarity*, *euclidean distance*, or *inner product* to determine closeness between vectors. Approximate Nearest Neighbor (ANN) algorithms are usually employed to optimize search speed and scalability, making it feasible to perform real-time retrieval in large datasets [35]. Applying this to the context of this thesis, similarity search allows for retrieval of relevant past experiences, such as previously encountered errors, by comparing the embedding of the current situation to those stored in the vector database. This works as a long-term memory, facilitating reuse of previous knowledge and more informed decision-making.

2.4 Retrieval-Augmented Generation

Building on the capability of vector databases to retrieve relevant information efficiently, Retrieval-Augmented Generation (RAG), originally proposed by Lewis et al. [36], integrates this retrieved external knowledge with LLMs to enhance their reasoning and generation abilities. By dynamically retrieving relevant documents or data points from relational or vector databases and incorporating them into the input prompt, RAG allows LLMs to go beyond the limitations of their fixed training data. This retrieval-augmented approach is particularly useful in complex scenarios where models need to adapt to new information or handle tasks requiring specialized knowledge.

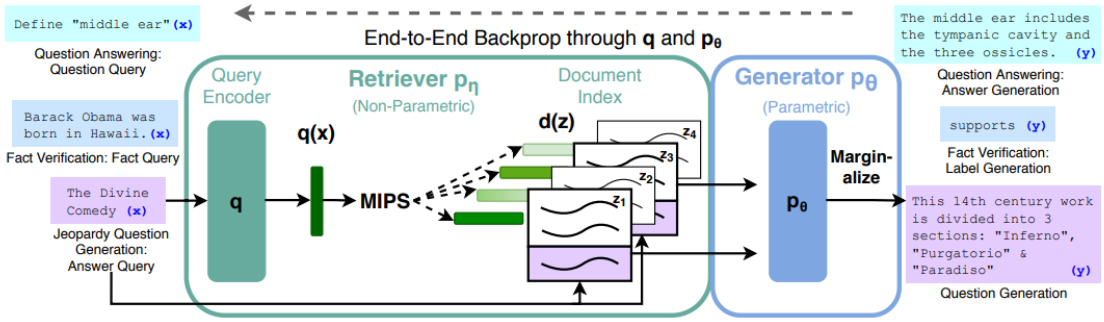

Figure 2.1: Illustration of the RAG architecture. The retriever searches an external document index to find relevant passages based on a query. These retrieved documents are then combined with the original query and passed to the generator, a sequence-to-sequence model, which produces a final output by conditioning on both the query and retrieved context. This approach enables the model to incorporate external knowledge dynamically during generation. Reproduced from Lewis et al. (2020) [36].

As illustrated in Figure 2.1 RAG architectures typically consists of two key compo-

nents, one retriever and one generator. The retriever searches an external knowledge source, e.g. a vector database, for relevant documents based on a query. The retrieved documents are then passed to the generator, usually a *transformer*-based LLM, which combines the retrieved content with an *in-context learning* prompt to generate more informed responses [37]. For instance, in response to a query such as *"Why did the braking system fail during the endurance test?"*, the retrieval component may identify and return semantically relevant events and error logs from a vector-based knowledge repository. The generator can then utilize an augmented and contextualized prompt during inference as follows:

```
Query: Why did the braking system fail during the endurance test?
Retrieved Context:
17:10 - Brake temperature exceeded threshold.
17:15 - Hydraulic fluid pressure dropped below recommended level.
17:30 - Noticing slight reduction in breaking power.
Answer:
```

This way, the language model is provided with additional context during queries, which can improve factual accuracy, reduce the risk of hallucinations, and support overall decision-making.

2.5 LLM Agents

With the concurrent advancements in LLM technology, the concept of LLM-based agents has recently gained significant traction. The idea behind an LLM agent is to use an LLM as the central component of an autonomous system specialized for solving specific tasks. These LLM agents are typically built around modular workflows, usually including components for planning, memory, and tool use, which enable more structured LLM behavior. One example of a possible agent architecture follows a perception–planning–action loop: the agent interprets a user query, decomposes it into steps, and executes them using tools that search the web, does API calls or query information from databases [38]. For example, an agent such as SciAgent [39], designed for scientific research, can summarize papers and generate hypotheses by sequentially invoking tools like literature search engines. These specialized components distinguish agents from generic LLM calls and enable them to perform complex, domain-specific tasks more effectively.

2.6 Prompting Techniques

As LLM agents rely on language models as their core reasoning and communication engine, prompting plays a critical role in shaping their behavior and capabilities. Carefully designed prompts serve as the primary mechanism for instructing agents, enabling them to perform specialized tasks, communicate effectively internally or with other agents, and maintain coherence across complex workflows. The following sections will introduce a number of prompting techniques that will be utilized to improve the performance of the agents in this thesis.

2.6.1 Role Prompting

Role prompting is a prompting technique in which LLMs are guided to behave in specific, predefined roles in order to better align their outputs with task-specific expectations. By explicitly assigning a role such as "data analyst," or "project manager" the model is encouraged to adopt a context-specific persona, which can significantly improve both the relevance and accuracy of its responses [40].

In LLM-based systems with several agents, each agent typically performs a specialized task, and role prompting ensures that the agent interprets its responsibilities correctly. For instance, assigning the role of a "summarizer" to one agent and a "validator" to another not only improves clarity but also reduces ambiguity in how each agent should process and respond to information. Recent work has shown that role prompting can both reduce hallucinations and improve consistency by providing clearer behavioral constraints and division of tasks [41].

2.6.2 In-Context Prompting

In-context prompting encompasses *zero-shot*, *one-shot*, and *few-shot* methods, all of which leverage an LLM's ability to learn from instructions and examples provided directly in the prompt. *One-shot prompting*, in particular, provides exactly one task-specific example alongside the query, unlike zero-shot prompting, which relies solely on natural language instructions, or *few-shot prompting*, which offers multiple examples to guide the model [18]. For instance, by embedding a single illustrative instance of the desired input–output behavior (for example, a sample of an error and its correction, or a snippet of dialogue), *one-shot* methods exploit the model's in-context learning to infer task structure and output format without explicit fine-tuning. This can be especially valuable in complex reasoning scenarios, where a single well-chosen example helps the agent align its internal reasoning and communication patterns to domain-specific conventions, yielding more accurate and context-aware outputs [42]. Although, the success of *in-context prompting* depends significantly on example selection, prompt formatting, and model size, with larger models generally demonstrating stronger performance [43].

2.6.3 Chain-of-Thought Prompting

Chain-of-Thought (CoT) prompting is a technique that encourages LLMs to generate intermediate reasoning steps before arriving at a final answer, thereby enhancing their problem-solving abilities [44]. This method mimics human cognitive processes, where individuals typically break down complex tasks into a sequence of smaller, manageable steps. By guiding the model to provide a step-by-step process in its output, CoT prompting improves the transparency and correctness of the model's reasoning, especially in tasks that require logical inference or multi-step reasoning [45]. Research has shown that CoT prompting significantly boosts performance on challenging benchmarks, such as arithmetic reasoning and inference based on common sense, by ensuring that the model does not jump to conclusions prematurely [43]. In systems incorporating several agents, CoT can also further enhance collaboration

by allowing agents to share their reasoning process, improving interpretability and coordination in decision-making.

2.7 Multi-Agent Systems

As single LLM-based agents continue to improve in reasoning and text generation, multi-agent systems have emerged as a way to further enhance collaboration and optimize task management among LLM agents. While single-agent models can effectively plan and execute tasks, they struggle with scalability and specialization as complexity increases [46]. MASs address this by assigning specialized roles to different agents, enabling more efficient problem-solving by dividing complex tasks into simpler subtasks, which then can be solved through communication and coordination.

Recent frameworks structure these systems around four key dimensions: profiling, environment, communication, and learning [46]. Agents are assigned roles through predefined, model-generated, or data-derived profiling methods, ensuring that each agent has a distinct function suited to its task. Their operation is influenced by the type of environment they interact with, which can range from sandbox simulations where agents experiment freely, to purely conversational contexts where reasoning emerges through dialogue. Effective communication plays a central role in coordination, where agents can debate, co-operate or compete with each other. To enhance performance and adaptability, agents refine their capabilities through techniques such as memory retrieval and self-evaluation, allowing them to improve over time and tackle increasingly complex problems.

Furthermore, the role of external tools in multi-agent systems has become increasingly critical, enabling agents to interact with external environments, retrieve information, and perform actions beyond their native capabilities. Frameworks such as LangChain [47] equip agents with modular tool-use, allowing seamless API integrations, database queries, and even autonomous web browsing. It also facilitates structured reasoning and agent orchestration through its chain-of-thought pipelines and agent-executor patterns, enabling more controllable and transparent interactions with external systems.

Building on this, LangGraph [48] introduces a graph-based orchestration model for agents, allowing developers to construct dynamic workflows in which agents are treated as nodes in a directed graph. This structure supports asynchronous interactions, decision branching, and memory retention across complex sequences of agent actions. LangGraph's event-driven architecture also makes it well-suited for adaptive, long-running tasks that require persistent coordination between agents.

2.8 Evaluation of Natural Language Generation

Evaluating the quality of text generated by multi-agent systems presents significant challenges, largely due to the subjective and context-dependent nature of language itself. Unlike deterministic or numerical outputs, language is flexible, often ambiguous,

and heavily influenced by individual interpretation and context. This makes consistent and objective assessment particularly difficult, as the same text may be judged very differently by different evaluators [49].

In addition to this, traditional automatic evaluation metrics such as BLEU, ROUGE, or METEOR rely on comparing the generated text to a predefined reference [11]. While effective in constrained tasks like machine translation or summarization, this approach becomes problematic in open-ended contexts such as report generation, where no definitive ground truth exists. Although human-written reports can serve as a high-quality reference, using it as a strict gold standard becomes limiting, since a system-generated report may be equally valid or even superior while emphasizing different aspects or presenting information in a different way [10].

Currently, there is still no universally accepted framework for evaluating generated reports. While human qualitative evaluation deals with many of the issues present in automatic metrics, it introduces its own problems with variability, subjectivity, and limited scalability [50, 51]. This absence of established evaluation frameworks complicates comparison across generative systems, particularly in dynamic and complex domains such as report generation. However, some automatic frameworks have emerged for evaluation of freely generated text. The following sections will present two such frameworks.

2.8.1 Question and Answering for Evaluation

Question Answering (QA) has emerged as an approach to evaluate NLG systems by assessing their ability to convey accurate and relevant information. Unlike traditional surface-level metrics such as BLEU or ROUGE, which primarily measure lexical overlap between generated and reference texts, QA-based evaluation focuses on semantic correctness and content [52, 53].

In this approach, a set of questions is automatically or manually generated from the source text or reference summary, targeting the key information and facts contained within. The generated text is then queried with these questions, and its answers are compared against the expected correct responses. This enables an evaluation of how well the generation captures the critical information and supports correct inference [54]. Several evaluation frameworks, including QuestEval [52] and QAGS [55], have implemented QA for summarization and other NLG tasks. These frameworks typically rely on pretrained question generation and answering models, often based on LLMs, to automate the QA evaluation pipeline and achieve scalable, reproducible results [53].

2.8.2 RAGAS Evaluation Framework

The Retrieval-Augmented Generation Assessment Suite (RAGAS) framework provides a comprehensive approach for evaluating RAG systems, focusing on the quality and accuracy of generated text grounded in retrieved knowledge [56]. RAGAS leverages LLMs to automatically assess how well a generated output aligns with both the query

and the supporting context (i.e retrieved documents), making it particularly suitable for complex tasks such as natural language generation and fact-based dialogue.

Most RAGAS metrics operate by comparing the generated response against the provided context and, where applicable, a trusted reference response. This LLM-based evaluation enables fact based judgment of the content's relevance, correctness, and informativeness beyond simple word-by-word overlap. The RAGAS metrics utilized for this thesis are *Faithfulness*, *Factual Correctness*, and *Summary Score*, which will be explained in further detail below.

`Faithfulness` measures the proportion of statements in the generated text that are supported by the provided context, ensuring the generation does not hallucinate unsupported facts. It is formally defined as:

$$\text{Faithfulness} = \frac{\text{Number of Supported Statements}}{\text{Total Number of Statements}} \tag{2.1}$$

`Factual Correctness` evaluates whether the factual claims in the generated output are true when compared against a trusted reference text, independent of the input context. This metric can be expressed as:

$$\text{Factual Correctness} = \frac{\text{Number of Factually Correct Claims}}{\text{Total Number of Claims}} \tag{2.2}$$

`Summary Score` quantifies how well the generated response captures the important information from the retrieved contexts. First, keyphrases are extracted from the context and converted into yes/no questions, all with "yes" as the correct answer. These questions are then posed to the summary, and the score is computed as the ratio of correctly answered questions:

$$\text{Summary Score} = \frac{\text{Correctly Answered Questions}}{\text{Total Number of Questions}} \tag{2.3}$$

By combining these LLM-driven metrics, RAGAS offers a multidimensional evaluation framework that balances factual grounding, truthfulness, and informativeness, enabling detailed analysis and robust comparison of RAG systems [56].

3

Methods

Building on the theory presented in the previous chapter, this chapter presents the method employed throughout this thesis. It begins with a detailed introduction of the dataset, followed by a description of the baseline single-LLM system which will be used for reference in the evaluation pipeline. Next, the architecture of the MAS itself is defined, with an overview and description of each node together with an explanation on how they are connected. Finally, the evaluation frameworks are presented.

3.1 Dataset

The dataset consists of multiple manually written logbooks originating from truck expeditions. Each entry spans several days of testing and is structured in tabular format, with each row containing information about timestamps, event descriptions, reported issues, proposed solutions, and occasionally also images that provide additional context. An example logbook with fabricated data is displayed in Table 3.1 below. Notably, while the logbooks are semi-structured, directly extracting the raw text would result in completely unstructured data, as there is no standardized structure in the event entries. With background in this, key data is extracted, parsed, and combined into higher-quality structured data, hereafter referred to as *artifacts*. These artifacts are essentially prepared text data inputs designed to provide additional context to the LLMs during inference. They are created from data stored in various databases, and the process of their construction is explained in further detail in later sections.

3.2 Databases

To provide the MAS with high-quality data, two databases were created. The first database is a *SQLITE* relational database used for storing structured data from the logbooks, whereas the second is a *Milvus* vector database focused on retrieving similar previous errors. These databases naturally vary in type and content and will be described further in the following sections.

Table 3.1: 100% Fabricated example of generic truck expedition logbook data.

Time	Event Description
06:00	Started engine after overnight idle; initial RPM spike noted, stabilized within 30 s.
06:30	Performed full pre-check inspection; all systems nominal.
07:00	Cold idle test, slight vibration in cabin detected.
07:30	Warm-up drive on flat track; coolant temp reached optimal range in 6 min.
08:00	Acceleration test from 0–60 km/h, noted 1.5 s delay in throttle response.
08:30	Brake test at 40 km/h, front-left brake pad exhibited uneven wear; flagged for replacement.
09:00	Suspension compression test over bumps; rear suspension performance within tolerance.
09:30	Fuel efficiency monitored over 5 km loop, consumption at 13.2 L/100 km (expected: < 12.5).
10:00	Gear shift responsiveness assessed, minor hesitation shifting from 2nd to 3rd gear.
10:30	Partial load test (50 %), no drift or instability detected.
11:00	Hill climb at 12 % grade, power delivery consistent; torque within expected curve.
11:30	Cooling system check; fan engaged late, caused brief temp spike; software update scheduled.
12:00	Lunch break.
12:30	In-cabin noise test at idle, result: 52 dB (pass, target < 55 dB).
13:00	Speed governor check, governed at 92 km/h as expected.
13:30	Lane change at 60 km/h, slight oversteer noted.
14:00	Turning radius validation, no issues; 11.2 m radius confirmed.
14:30	Rough terrain traversal; rattling from dash panel; loose fastener corrected.
15:00	Brake retest; pad replaced earlier now performing within spec.
15:30	End-of-day system scan; no persistent error codes.

3.2.1 Relational Database

The primary objective of the relational database is to store contextual information from the testing logbooks. The database contains three tables: `sections`, `images`, and `section_images`, which is a mapping that associates each image with its corresponding section using the `section_tag` and `image_id` from the two other tables. The exact structure of the database is displayed in Figure 3.1 below.

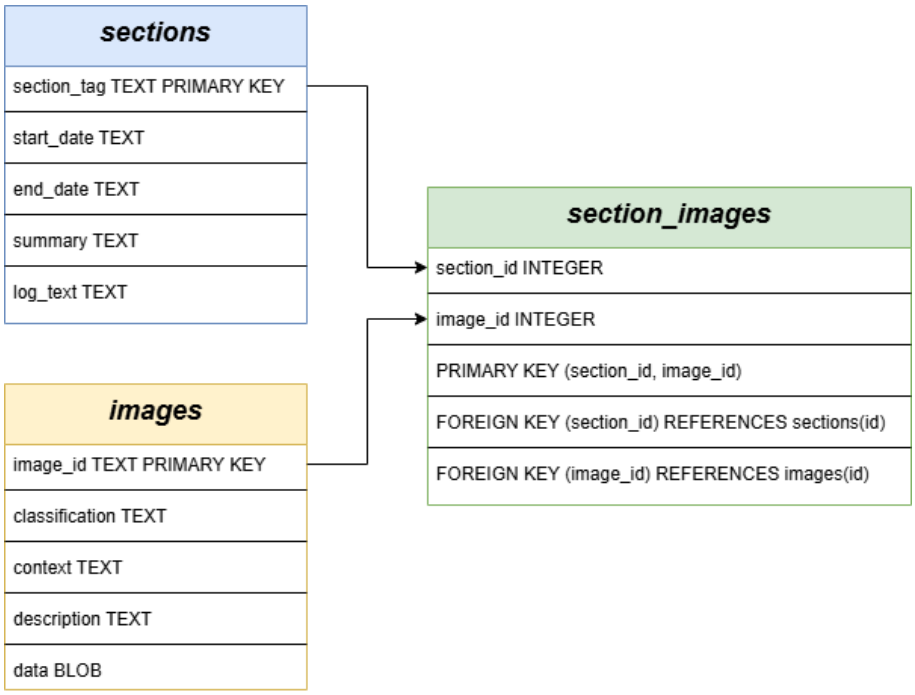

Figure 3.1: Structure of the Relational Database utilized in the MAS, including table names and attributes.

The `Sections` table contains information about each segment of the expedition, with each section representing a distinct part of the truck test. For each section, five separate attributes are stored: a section ID, the start and end dates of the test, a summary of key events generated by an LLM, and the raw logbook data.

Similarly, the `Images` table stores five attributes related to each image: an image ID, a description of the image content generated by a vision-compatible LLM, and the image data itself. Along with the `description`, each image is also assigned a classification, either *graph*, *table*, *map*, or *other*. Additionally, the context of the image is stored, meaning the textual information that appears in the same cell as the image in the logbook.

The `section_images` table does not contain any standalone data; it simply maps sections to their associated images.

3.2.2 Vector Database

The Milvus vector database is dedicated to storing specific errors and linking them to potential solutions. Because logbook entries are unstructured and can record an error at one timestamp and its resolution at another without any standardized mapping, a LLM is used to extract and associate error–solution pairs.

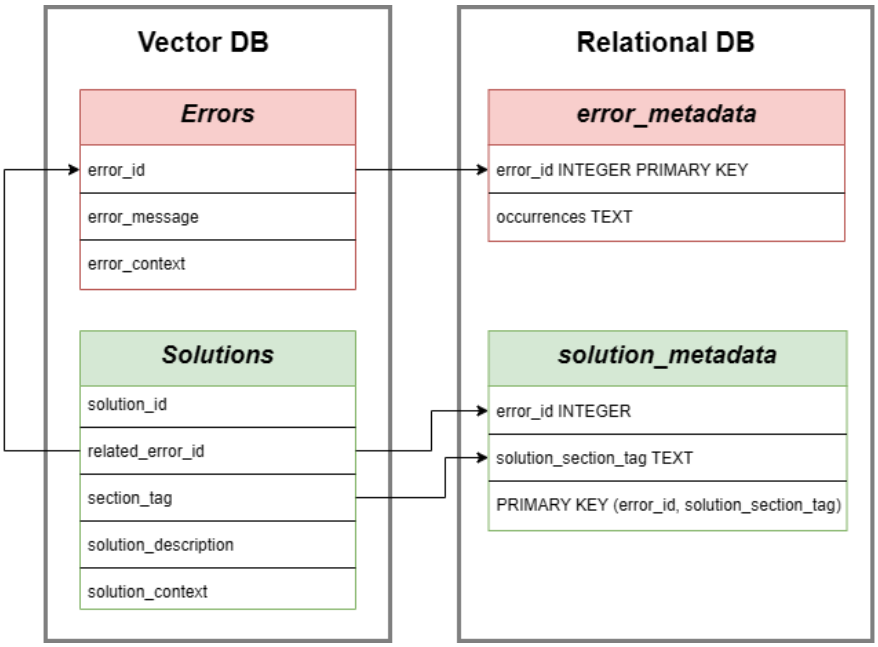

Figure 3.2: Structure of the Vector Database, including the errors and solutions partitions alongside the relational database that stores each vector's additional metadata.

The database is built with two distinct partitions under a shared schema, one for errors and the other for solutions, with partition meaning a logical grouping within the vector store that keeps error-related vectors separate from solution-related ones. Each partition is complemented by its own relational database, responsible for storing additional metadata associated with their respective entries. As each log file is processed, the vector database is incrementally updated by passing both the `error_message` and its corresponding `solution_description` through the *bge-m3* embedding model, which tokenizes the text and produces fixed-length dense vectors. These vectors are normalized to enable more efficient Inner Product (IP) similarity search, compared to directly computing cosine similarity, as it avoids the need for runtime normalization. The resulting embeddings are then appended to the appropriate partition along with their relational metadata.

Within the error partition, each entry tracks an `error_id`, an `error_message`, and any available `error_context`. Metadata such as the number of times a specific error has occurred is maintained in the relational database and linked via the corresponding `error_id`. This allows for the monitoring of historical error frequency.

Solutions are instead dependent on errors and cannot exist on their own. Each solution is therefore linked to a specific error through a `related_error_id` and assigned

a unique `solution_id` for identification and traceability. Contextual details like `section_tag`, `solution_description`, and `solution_context` are stored alongside the embedded vector in the solution partition. Meanwhile, the relational database tracks processed solutions during insertion to prevent duplicate entries.

By embedding errors and solutions, the database allows for the retrieval of similar vectors when a new one arise during inference. This enables the system to identify past errors and their solutions, offering similar contexts previously encountered together with potential solutions, even if not explicitly provided in the current logbook. The operational details of how the vector database is queried and updated are further explained in Section 3.5.4.3 and Section 3.5.4.8, respectively.

3.2.3 Acronym Dictionary

In addition to the databases, the system also incorporates a domain-specific dictionary that maps key concepts to their corresponding acronyms. This dictionary serves as a ground truth reference for interpreting and translating acronyms found in the logbooks. It is provided to the report-generating agents as an additional artifact to ensure accurate comprehension of domain-specific terminology and to maintain consistent and correct usage of acronyms throughout the generated reports.

3.3 System Architectures

The following sections describe the two LLM based report generation systems, the `Baseline` is presented in Section 3.4 and the `MAS` in Section 3.5. The primary differences between the systems revolve around the structure of the prompts and the input data. In the multi-agent setup, the overall prompt is divided into subprompts with more specific and specialized instructions for each agent. Additionally, each section of the multi-agent written report is generated using distinct artifacts, whereas the baseline system relies solely on the unstructured logbooks and one general prompt when generating the report.

3.4 Baseline System Overview

The baseline report generator is implemented using a single-agent architecture, with the main purpose of providing a way to compare the quality between a single- and multi-agent system. The baseline agent is provided with a simple role prompt with similar specifications as the multi-agent system. However, it is presented in one singular, and thereby less distinct prompt. This includes specifications on how to generate the report including headers, citations and general language. It also describes the input data and provides a one-shot example demonstrating how the data can be incorporated into a section. Lastly, the baseline model processes raw text logbooks as input, without the use of any structured or preprocessed *artifacts*. The exact prompt can be found in Appendix B.

3.5 Multi Agent System Overview

The multi-agent report generation system is instead structured around two main planning agents: the `Data Extraction Planner` and the `Writing Planner Agent`. These agents coordinate a pipeline composed of multiple functional nodes, where each node performs a specialized subtask. The system maintains a shared internal state, provided by `State` class from the `LangGraph` library [57]. This state is updated at each step and is used both to route decisions and to augment prompts for improving the context provided to the agents.

3.5.1 Component Overview

This section will briefly describe the main components of the system.

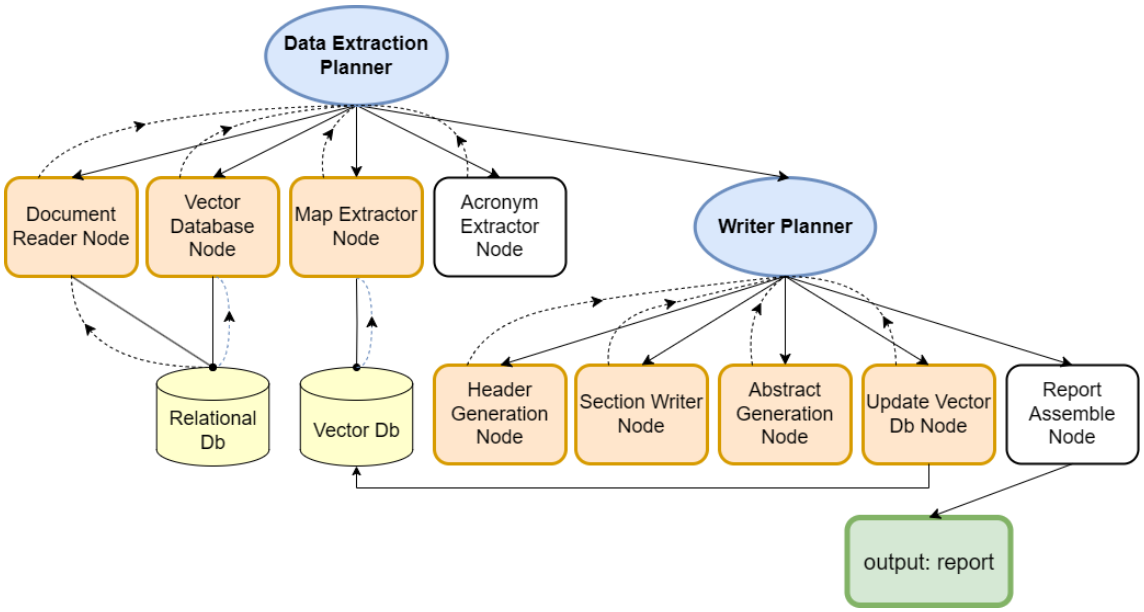

Figure 3.3: Overview of the complete multi-agent system architecture. Databases are shown in yellow, while nodes incorporating LLMs are orange. Oval components denote *planning*-nodes, while rounded-rectangular components denote *worker*-nodes. Detailed descriptions of each node's functionality are provided in Section 3.5.4.

- `Data Extraction Planner`: Decides which data extraction node to activate next. It routes the retrieval of all necessary information and corresponds to the *retrieval* phase in the RAG framework. Its purpose is to gather relevant artifacts and metadata, such as similar past errors and associated solutions, before report generation begins. It builds upon the `Phi-4` model due to its fast and efficient inference.
- `Extraction Nodes`: Includes the *Document Reader Node*, *Vector Database Node*, *Map Extractor Node*, and *Acronym Extractor Node*. These *worker*-nodes query the databases to collect information based on their respective area. Retrieved artifacts are stored in the system state and used to augment future

prompts. These nodes utilize the `Llama-33-70b` for it's strengths in instruction following in well defined tasks such as retrieval.

- `Writing Planner`: Once data collection is complete, this agent takes over and coordinates all steps in report generation. It determines the sequence of writing operations and corresponds to the *generation* phase in the RAG framework. This node also builds on the `Phi-4` model for fast and efficient routing.

- `Writing Nodes`: Includes the *Header Generator Node*, *Section Writer Node*, *Abstract Generator Node*, *Update Vector Db Node* and Report Assembler. These generative *worker*-nodes receive prompts enriched with previously retrieved artifacts to improve relevance and accuracy of the generation process. These nodes build upon the `GPT-4o` model, for it's strengths in advanced writing and ability to keep coherence over long documents.

3.5.2 Prompt Data Mapping

Engineering reports typically follow a consistent overall format, but the internal structure and content within each section can vary significantly. To adjust for this, the MAS employ a method hereafter referred to as `prompt data mapping`. This approach uses a dictionary that defines which data inputs are passed to each part of the report during generation, which provides the system with precise control over the content. It also enables the system to flexibly adapt section-level prompts, customize input formatting, and dynamically adjust headings or phrasing based on the type and structure of the data provided.

For example, certain headers in the report always need to be present and always use the same wording, such as *Introduction*, *Purpose and Objectives*, or *Vehicle Technical Specifications*. These are mapped as fixed headers, and no generation is allowed for them. They appear in the same format every time, regardless of the content. Other sections, however, depend the *artifacts* extracted from the test. For example, under a header like *Vehicle Issues Found During Test Run*, the subheaders depend entirely on the types of problems identified. In one run, *Electrical Faults* and *Issues with Overheating* might be detected, while other runs only encounters *Mechanical Failures*. The subheaders for *Vehicle Issues Found During Test Run* therefore needs to be generated dynamically using the available data.

However, the section texts that follow these headers cannot be generated using a single, uniform prompt or dataset. Each section serves a distinct purpose and requires different types of input information and instructions to produce relevant and coherent content. For example, the *Introduction* section may need high-level contextual data, while *Vehicle Technical Specifications* relies on structured tabular input. To accommodate this, `prompt data mapping` not only controls the structure of the report but also ensures that relevant data and prompt instructions is routed to each section reflecting its content.

3.5.3 Routing and Planning Logic

Routing logic was implemented to address the MAS coordination challenge, particularly the risk of conversation memory loss, by ensuring that each planning component receives the precise contextual information required to maintain a coherent, shared task understanding. Both the `Data Extraction Planner` and the `Writing Planner Agent` are equipped with the same routing logic to decide which node to pass the state to next. This decision is based on a combination of current system state, tool history, and past outcomes stored in memory.

Two memory structures are used to guide these decisions:

- `Scratchpad`: Keeps track of which tools that has been used and prevents redundant queries. Tools may only be invoked twice and cannot be called again with the same query. This limit reflects that repeated failures usually signify data related problems rather than system faults, and further retries would impose unnecessary computational overhead.
- `Chat Memory`: Functions as a shared history of task completions. Each node appends a message indicating whether a task succeeded or failed, which the planner then uses to determine if any nodes need to be revisited.

At each step, the planner receives a system prompt reflecting the current internal state and responds with a routing decision in structured JSON format. This is done using `with_structured_output`, which enforces valid output and forces the model to choose one of the predefined worker nodes and specify the task to perform [58]. This ensures unambiguous control flow and consistent execution across planning steps

Listing 3.1: Example Routing Format

```
{
  'node': "<next_node>",
  'current_query': "<task_description>"
}
```

This mechanism keeps the system moving forward in a structured way while still allowing flexibility if tasks need to be re-tried or skipped. Importantly, the architecture decomposes responsibilities: Only the *planning*-nodes are contex-aware, leveraging the full `scratchpad` and `chat history` to make context-sensitive routing decisions. In contrast, the remaining *worker*-nodes are pure executors, each responsible for a narrowly defined subtask, simply reporting success or failure. This clear separation both simplifies individual node design and strengthens overall inter-agent alignment. The full prompt structures and routing logic for both planners are provided in Appendix A.1 and Appendix A.2.

3.5.4 Detailed Worker Node Descriptions

With the system's high-level architecture and data management strategies now outlined, the following subsections present a comprehensive explanation of each

worker node's functionality. Both extraction and writing nodes are described, including how they contribute to the report pipeline.

3.5.4.1 Document Reader Node

The `Document Reader` parses log files section by section to extract high-level summaries, test purpose and objectives, specifications, and images. Each image is also analyzed using a vision compatible LLM model, namely GPT-4o, to extract its content along with a classification of what the images contains. The node additionally identifies new errors and corresponding solutions using an LLM-based extraction pipeline. Each extracted artifact is tagged and appended to the internal state. The prompt used for extracting error-solution pairs can be found in Appendix A.3.

3.5.4.2 Map Extractor Node

The `Map Extractor Node` begins by querying the relational database's `images` and `section_images` tables to retrieve every image classified as a *map* along with its binary data, caption text, and associated `section_tag`. Once retrieved, each map image and its associated context are submitted a second time to an image-based LLM, *gpt-4o*, but now configured for `with_structured_output`. From each map, the agent then extracts the following information:

- **Location:** area name, route description, and latitude/longitude coordinates
- **Measures:** quantitative metrics such as total distance, elevation gain, and duration
- **Description:** a concise (20–30 word) figure caption summarizing the map's contents

These fields, together with the original `section_id` and `image_id`, are combined into a JSON object and appended to the state's `artifacts_maps` array.

3.5.4.3 Vector Database Node

The `Vector DB Node` utilizes the vector database described in Section 3.2.2 to provide contextual artifacts that link newly encountered errors to semantically similar past issues and their corresponding solutions. For each new error, the node first computes an embedding of the error message using the same *bge-m3* model employed during database construction. This embedding is then used to query the vector index using IP similarity, retrieving the top-10 most similar historical error entries.

Rather than applying an absolute distance cutoff, the node ranks candidates by IP score, which effectively serves as a proxy for semantic similarity due to the normalization of embeddings during both indexing and querying of the vector database. Each retrieved candidate includes its original error message, associated context, the number of past occurrences, and any linked solutions.

The node initially selects the top candidate with the highest similarity score. If this top match lacks associated solutions, it scans the remaining candidates for the nearest

solved error whose score is within a relative threshold of 0.1 from the best match. The threshold is empirically chosen as a balance: low enough to ensure semantic closeness, but high enough to allow fallback matches that still carry relevant context. This prioritization ensures that past errors with associated solutions are preferred, as they provide directly actionable guidance for resolving the new issue. If no solved case satisfies this condition however, the original top unsolved match is retained.

Finally the selected error, regardless of whether it includes a solution, is attached to the current error entry, enriching it with historical context. This structured artifact enables downstream components, such as the `Section Writer`, to draw informed comparisons and generate more accurate and context-aware outputs.

A fabricated example of a newly encountered error before enrichment is shown in Listing 3.2, where no contextual information or solutions are initially available.

Listing 3.2: Example of a New Error Artifact before processing by the Vector DB Node

```
{
  'new_error_message': "Steering vibrates intermittently at low speed.",
  'new_error_context': null,
  'new_solutions': []
}
```

After processing by the `Vector DB Node`, the error artifact is enhanced with a semantically similar previous error, complete with context, solutions, and occurrence metadata, as illustrated in Listing 3.3.

Listing 3.3: Enriched Error Artifact after `Vector DB Node` processing

```
{
  'new_error_message': "Steering vibrates intermittently at low speed.",
  'new_error_context': null,
  'new_solutions': [],
  'most_similar_previous_error': {
    'previous_error_message': "Steering shimmy under 15 km/h.",
    'previous_error_context': "Checked tie rod ends and wheel alignment.",
    'previous_solutions': [
      {
        'solution_description': "Adjusted wheel alignment and replaced worn tie
            rod.",
        'solution_context': "Vibration eliminated after alignment correction.",
        'section_tag': "S1589"
      }
    ],
    'number_of_occurrences': 2,
    'occurrences': [
      {
        'section_tag': "S1589",
        'error_date': "2024-07-10 09:15"
      },
      {
        'section_tag': "S1620",
        'error_date': "2024-08-02 14:45"
      }
    ]
  }
}
```

3.5.4.4 Acronym Extractor Node

In addition to the vector and raw-text artifacts, the system builds an `artifacts acronyms` dictionary by matching against the ground-truth *Acronym Dictionary* provided as an auxiliary artifact. For each document, the `Acronym Extractor` scans the full raw text and applies two simple, case-insensitive regular-expression tests for each known acronym-explanation pair:

1. It searches for the acronym key itself as a whole word, using the pattern:

$$\texttt{pattern\_key} = \texttt{\textbackslash b} + \texttt{re.escape(acronym)} + \texttt{\textbackslash b}.$$

2. It also searches for the human-readable explanation string anywhere in the text, via `re.escape(explanation)`.

New acronyms are recorded in the `State` and a summary of how many new entries were found is recorded in `chat_history`. This ensures that downstream *generation*-components such as the `Section Writer` always have a complete, up-to-date map of domain-specific acronyms for the current test trial, improving both the accuracy of in-text expansions and technical domain knowledge during generation.

3.5.4.5 Header Generator Node

The `Header Generator` creates a nested report outline using both fixed templates and dynamic generation. It utilizes prompt context and artifact mapping to decide which subheaders to include. Dynamic generation for headers can be enabled or disabled using the `generate_subheaders` flag within `prompt_data_mapping`.

In addition, the `artifacts_to_partition` field in `prompt_data_mapping` enables segmentation of artifacts to ensure that extracted information is correctly routed to the appropriate subheaders. This becomes particularly important when single errors from the artifact contains information relevant to multiple subheaders. In such cases, ambiguity can arise if the artifact is not properly segmented, leading to repeated or misplaced content where unrelated issues are discussed under the wrong headings. Partitioning addresses this by explicitly dividing the artifact's content, ensuring each subheader receives only the relevant data. This reduces redundancy, minimizes the risk of hallucinations, and ensures that each type of issue is accurately presented within its intended context. The partitioning itself is handled by a helper function that calls an LLM to split and assign artifacts to the headers it finds most relevant (see Appendix A.4 for details).

3.5.4.6 Section Writer Node

Once the nested outline is generated by the `Header Generator node`, the `Section Writer` traverses the header structure recursively, generating the content for each section in sequence. For each header and subheader generated by the Header Generator, only the artifact data and the context associated with that node, as mapped in `prompt_data_mapping`, are utilized during the generation process. After writing a section, the text is fed back in as `memory` to enable later sections to build

upon what has already been written. This lets the model maintain consistency and avoid repeating or contradicting itself. The writer prompt also tracks `used_acronyms` through the internal state, ensuring that once an acronym has been defined, only its short form is used in the rest of the report.

Below is the exact prompt template used by `section_writer_tool`. Each placeholder is filled in for the current header during recursion:

Listing 3.4: Section Writer Prompt, incuding detailed agent descriptions and specification of the data recovered from the RAG retriever.

```
'prompt' = (
    "### Section Writer Request\n"
    "Please ensure you fully understand the concepts and the technical
        abbreviations provided below before proceeding. Think step by step.\n\n"

    "### INSTRUCTIONS:\n"
    "You are a scientific expert report writer responsible for composing
        individual sections of a comprehensive report that is going to be
        published at a conference.\n"
    "Your task is to write the section for the given **Header:**. To do so, use
        the provided **Data:** as a guiding resource, and carefully review the
        following **Acronym and Explanations** to understand all technical terms
        in context.\n"
    "Also, take into account any **Prior Sections:** for consistency and to avoid
        redundancy. Use **Used Acronym** to check if an abbreviation has already
        been defined and explained.\n"
    "**Additional Context:**\n"
    "\\\"{prompt_context}"\\\"

    "### IMPORTANT:\n"
    "- Only produce a section if the provided content contains sufficient **Data
        :**.\n"
    "- Do not invent or hallucinate any information.\n"
    "- If insufficient **Data:** is available, output ONLY and EXACTLY: 'Not
        sufficient data for producing this section'.\n"
    "- Do not repeat the header (avoid starting with the header text or markdown
        headings).\n"
    "- Use succinct, clear, and formal language, ensuring every word is purposeful
        .\n"
    "- STRICTLY adhere to **Used Abbreviations**: if an abbreviation has been
        defined earlier, do not include its explanation again; only the short form
         must appear in all subsequent sections.\n\n"

    "========================\n\n"
    '**Header:**' "\\\"{header}"\\\"
    '**Abbreviations and Explanations:**' "\\\"{abbreviations}"\\\"
    '**Used Abbreviations:**' "\\\"{used_abbr}"\\\"
    '**Data:**' "\\\"{artifact}"\\\"
    '**Prior Sections:**' "\\\"{memory_text}"\\\"
    "### Section Text:\n"
)
```

The main components are:

- **Header:** The title of the current section.
- **Data (artifact):** JSON-formatted error or image entries relevant to this header.
- **Acronyms and Explanations:** Full definitions for any technical terms.
- **Used Acronyms:** A running list of acronyms already expanded, so subsequent sections only use the short form.

- **Prior Sections (memory):** All previously generated text in this branch of recursion, allowing the LLM to stay on-topic and avoid contradictions.
- **Prompt Context (optional):** Extra instructions for specific sections, e.g. enforcing correct citation format, linguistic tone adjustments, or emphasizing technical depth.

The optional `prompt_context` field in the `prompt data mapping` is what enables specific subsections to receive extra guidance through targeted prompt instructions during generation. For example, if writing the "Electrical Faults" subsection under "Vehicle Issues Found During Test Run," the `prompt_context` might be constructed as shown in Listing 3.5. In this context, placeholders such as `«CITATION:section_tag, error_date»` and `«IMAGE:image_id (caption:concise_caption)»` instruct the `Section Writer` to insert citations and visual references appropriately during inference. Later, these markers can then be detected by the `Report Assembler Node` via regex and replaced with the correct figure numbers, captions, source references, and hyperlinks during post-processing.

Listing 3.5: Example `prompt_context`

```
'generated_prompt_context': (
  "It is very important that you only use the parts of **Data** relevant to this
      **Header**. "
  "The **Data** is a JSON object combining error logs and image entries. "
  "For each error entry, insert citation markers in the format:\n"
  <<CITATION: section_tag, error_date>>
  "And for each graph image, insert:\n"
  <<IMAGE: image_id (caption: concise_caption)>>
  "Only include markers when they match the current topic, and never reuse the
      same image twice."
)
```

3.5.4.7 Abstract Generator Node

Once all sections have been written, the `Abstract Generator` produces a concise summary using the full document as context. This input consists of the previously generated content by the MAS, passed along from earlier stages. Drawing from internal memory and prior text, the generator ensures the abstract captures key themes, results, and conclusions, without duplicating headers or including images.

3.5.4.8 Update Vector Database Node

Before the final report is generated, the vector database is updated at the end of each execution with newly observed error-solution pairs to continuously improve retrieval accuracy and generalization across future test trials. This ensures that even previously unencountered but semantically similar issues can benefit from accumulated context and historical solutions.

The update process begins by embedding each newly extracted `error_message` and its corresponding `solution_description` using the same embedding setup described in Section 3.2.2. These embeddings are then used to check for semantic duplicates based on vector similarity. However, rather than inserting all new entries blindly, a

multi-stage filtering pipeline is employed to avoid duplicate entries, as illustrated in Figure 3.4:

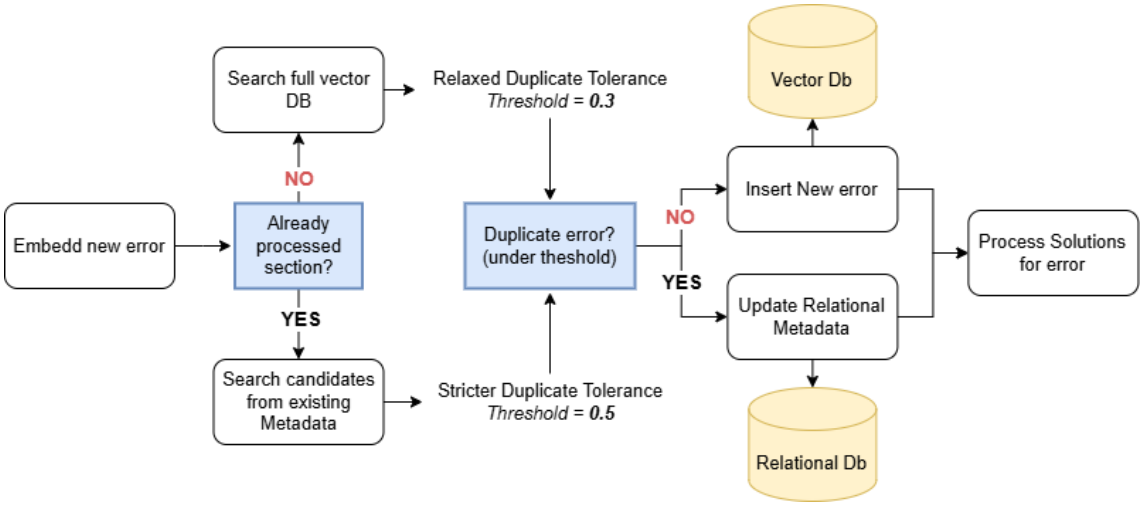

Figure 3.4: Overview of the Vector DB update pipeline.

1. The first step of the pipeline checks whether the current `section_tag` has already been processed. If so, pre-existing error metadata is queried to narrow the candidate set. Specifically, only errors that have previously occurred in the same section are considered, reducing the number of comparisons needed during vector similarity search. This step is particularly useful when the same files are reprocessed, helping to avoid reinserting known entries and improving search efficiency.

2. If no entries exist for the current section, the system proceeds with a full vector database search, querying across all previously stored embeddings to identify potentially similar errors. This fallback is necessary when encountering new or previously unprocessed content.

3. Regardless of whether the candidates are retrieved via metadata filtering or a full vector search, semantic similarity is assessed using the inner product (dot product) between embedding vectors $\vec{a}$ and $\vec{b}$:

$$\text{inner\_product}(\vec{a}, \vec{b}) = \vec{a} \cdot \vec{b} \tag{3.1}$$

$$\text{cosine\_distance} = 1 - \vec{a} \cdot \vec{b} \tag{3.2}$$

   Since the embeddings are normalized prior to insertion and search, the inner product effectively becomes cosine similarity. Therefore, lower cosine distances (computed as $1 - $ `inner_product`) indicate higher semantic similarity.

4. To determine whether the error should be treated as a duplicate, a similarity threshold is applied to the computed cosine distance. This threshold varies depending on the search strategy:

   - For metadata-based searches, a stricter duplicate threshold (e.g., cosine distance < 0.5) is used. This assumes the section has already been

processed and that we are likely re-encountering the same underlying error. However, since LLMs are utilized to extract errors from unstructured log data and are inherently probabilistic, the same issue may be phrased differently across runs. Therefore this threshold allows for moderate semantic variation, ensuring that equivalent errors are still recognized as duplicates, while genuinely distinct errors are allowed to pass through.

- For full vector database searches, a stricter duplicate threshold (e.g., cosine distance < 0.3) is applied. This path assumes new content is being processed, so the threshold is set conservatively to avoid incorrectly filtering out novel but valuable errors. Only highly similar entries are treated as duplicates, allowing moderately similar ones to be inserted as new entries. This helps extend the database with more diverse and distinct errors, improving the precision and coverage of future similarity-based retrieval.

5. If the cosine distance between the new error and the top match falls below the appropriate threshold, the error is treated as a semantic duplicate. In such cases, no new vector is added. However, the associated metadata is still updated in the relational database to record that the error has occurred again, including the current `error_date` and `section_tag`.
6. If no sufficiently similar vector is found (i.e., the cosine distance falls outside the defined threshold), the error is treated as novel. It is then inserted into the vector partition and initialized in the metadata store as a new entry

The same update logic extends to solutions, but with an important distinction: all extracted solutions are processed, regardless of whether their associated error is new or already known. This is crucial because even when an error is recognized as a duplicate and thus not re-inserted into the vector database, it may still be accompanied by a solution that has not been previously recorded. In such cases, the new solution is instead appended to the existing error entry, allowing the system to accumulate multiple resolutions for a single error over time.

To determine whether a solution is truly novel, each one is first linked to its corresponding `error_id` and checked against the solution metadata, which maintains a record of all `section_tag`s previously associated with that error. If the current solution has already been logged for the same section, it is skipped to avoid duplicate entries. Otherwise, it is embedded and compared against existing solution vectors linked to the same error using a stricter cosine distance threshold (e.g., < 0.2). If no close match is found, the solution is inserted into the vector partition and its metadata is updated.

3.5.4.9 Report Assembler Node

Once all sections have been written and the vector database has been updated, the `State` is passed to the final node of the MAS network, the `Report Assembler`. This node formats the final report into a document. It creates a title page, applies styling, inserts tables of contents and figures, embeds images, and hyperlinks references. It

also pre-processes any citation markers and ensures all figures and tables are included, numbered and listed appropriately.

3.6 Evaluation

Evaluating NLG systems presents a fundamental challenge in open-ended generation tasks due to the absence of a single ground truth [11]. Since a given text can be summarized or written in multiple valid ways, there is no definitive reference output. As a result, human evaluation has become the gold standard for most NLG tasks. However, while reliable, human evaluation is both costly and time-consuming.

In this work, we aim to leverage both the reliability of human evaluation and the scalability of automated NLG metrics to assess the quality of the report-generating agent system. By combining these two evaluation strategies, we aim to examine how well an automated approach compares to the quality of the human evaluation gold standard.

To perform the evaluation, we consider three following reports:

- **LLM-Baseline:** A report generated by a singular LLM agent in one step from a single prompt.
- **LLM-MA-report:** A report produced by the proposed multi-agent system.
- **Human-written report:** A report written by a human based on the same task description and input logbook data.

In addition to this, a complementary assessment is conducted to isolate and examine the effects of the different prompting techniques presented in chapter 2. This is done with an evaluation pipeline that targets a single section within the analysis chapter of the report. The section is generated four times using different prompts: the full `Section Writer` prompt, along with three ablated variants in which CoT reasoning, RAG, and *one-shot* prompting are each individually removed. All versions are generated using the same input data to ensure consistency, with the goal of providing a comparison of how each prompting technique influences the factuality and content of the written section. The actual evaluation score for this part is calculated using three automated RAGAS metrics, namely faithfulness, factual correctness and summary score.

3.6.1 Human Evaluation

The first component of the evaluation pipeline is based on human evaluators who qualitatively assess the set of reports presented in the previous section. These evaluators are Volvo engineers with varying degree of familiarity with engineering reports. To structure the evaluation process, we use five predefined metrics grounded in the ideas presented by Celikyilmaz et al [49]. The metrics, together with their respective definitions are displayed in the table below:

Table 3.2: Definitions of the evaluation metrics used by the evaluators.

Metric	Description & Evaluation Guidelines
Fluency	This metric assesses the general language in the report. Consider: • How appropriate, grammatically correct, and readable is the language in the context of a technical report?
Usefulness	This metric assesses whether the reader can easily understand the test and benefit from the report. Consider: • Did you understand/learn anything from the test by reading the report? • How interpretable were the insights in the report?
Repetition	This metric assesses how much the report repeats itself within and across sections. Consider: • How often does the report repeat itself? • Is there any redundant information?
Factuality	This metric assesses how well the report is correct and based on facts*. Consider: • Is there any information that is incorrect or contradictory? • Are the technical terms and specifications accurate? **If you don't know, leave this question blank.*
Structure	This metric assesses the overall structure of the report. Consider: • How well does the report follow a technical report structure? • How is the information structured/presented in each section?

These metrics should capture the most important aspects for assessing the quality of a generated text. Evaluations are conducted through a survey in which respondents are presented with one report at a time, in randomized order. For each report, one metric is evaluated at a time. Each metric is first defined in the context of the specific reports, after which the evaluator rates it on a Likert Scale, i.e. a discrete scale from 1 to 7, and provide a brief explanation with examples for the score.

3.6.1.1 Minimizing Bias in Survey

To ensure the validity and reliability of collected responses, the survey design was carefully constructed following best practices in questionnaire methodology, as outlined by Boynton and Greenhalgh [59]. Particular attention was given to reducing common forms of response bias that can compromise the quality and interpretability of survey data.

First, to avoid the introduction of bias through double-barreled questions, where two or more distinct concepts are embedded in a single item, the survey was designed such that each question targeted only one metric at a time. This approach allowed respondents to focus on a single evaluative criterion, reducing the mental burden and minimizing the risk of respondents mixing up responses between questions.

Second, while short questions can reduce respondent fatigue, overly concise phrasing may strip away essential context needed for accurate interpretation. As Boynton and Greenhalgh emphasize [59], clarity and context are vital for reliable data collection. To support insightful responses, each metric includes a concise explanation of the metric being assessed. These introductions helps standardize respondent understanding and provid necessary framing before evaluating the quality of each metric.

Lastly, to mitigate issues related to forced or imprecise scoring, the survey employed a discrete seven point scale. The selected scale provides a constant interval that provides sufficient granularity to capture nuanced opinions, while also including a neutral midpoint. This design reduces the likelihood of respondents feeling pressured to choose between inadequate alternatives and improves the interpretability and comparability of the collected data.

3.6.2 Automated Evaluation Framework

The automated evaluation framework in this thesis is inspired by G-EVAL, introduced by Yang et al., which has shown higher alignment with human judgment than commonly used automatic metrics such as BLEU, ROUGE, and METEOR, particularly for open-ended text generation tasks [11]. Unlike traditional metrics that depend on predefined reference outputs, this framework uses a LLM to evaluate text quality in a more flexible and context-aware manner.

This thesis adopts a similar approach by prompting the LLM to evaluate each generated report using the same five criteria as in the human assessment: fluency, usefulness, repetition, factuality, and structure. The model is guided through a step-by-step reasoning process to assign ratings on a seven-point scale, mirroring the evaluations made by human experts. A comparison between human and automated evaluation results is presented in Section 5.4, to validate the method's alignment with the reference of expert judgment.

3.6.2.1 Setup Automated Evaluation Framework

Inspired by recent advances in language model self-evaluation, a multi-round reflective review setup similar to the AI Scientist framework is adopted [60]. For each report

and every evaluation metric, a *GPT-4o* based evaluator simulates a Multi-Agent review process, where the model builds on its own previous reasoning. However, instead of traditional self-reflection, a modification is introduced, motivated by recent findings in the Multi-Agent Debate and Self-Consistency (MADS) framework [61]. While reflection-style methods aim to iteratively refine reasoning, they often suffer from a Degeneration-of-Thought (DoT) problem: once the model becomes confident in its initial belief, it tends to reinforce that view across subsequent steps, even when it is incorrect. This limits the diversity and critical assessment of the evaluation.

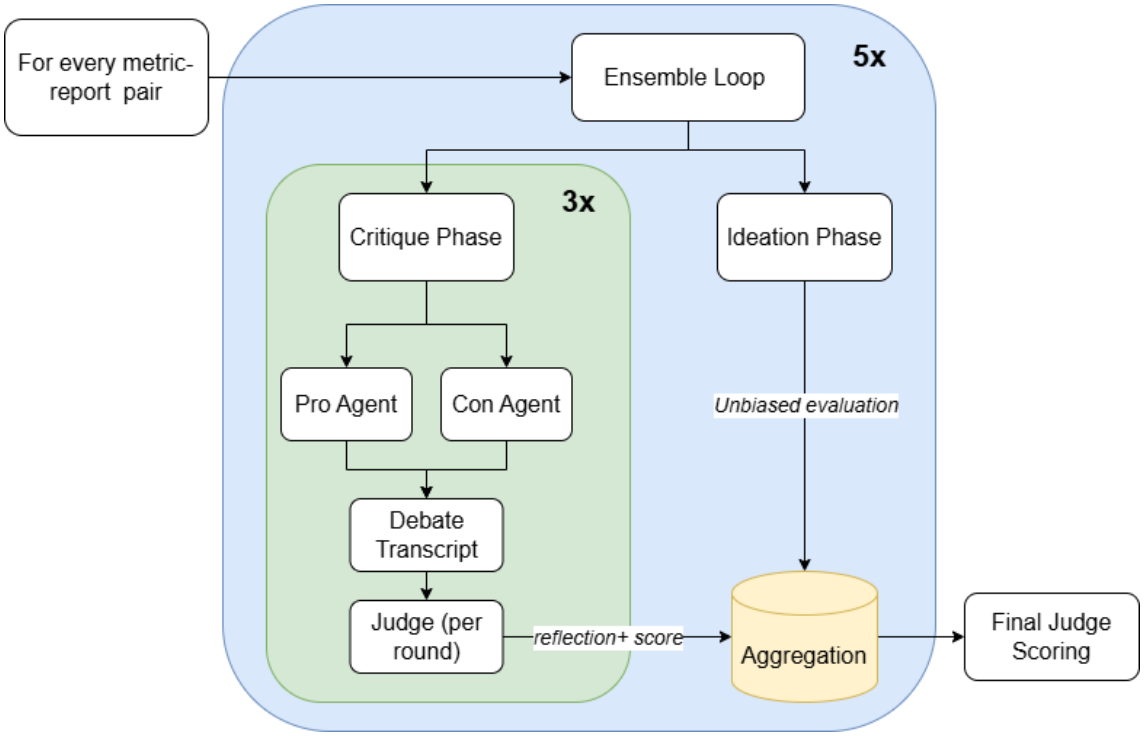

Figure 3.5: Overview of the Automated Evaluation pipeline.

To address this issue, the self-reflection rounds are replaced with MADS-style argumentative critique. An overview of this multi-phase evaluation setup is shown in Figure 3.5. Similar to the human evaluation, each report- and metric-pair is evaluated in an ensemble of five iterations. In each iteration, the evaluation process begins with an *unbiased ideation phase*, where the model performs an initial standalone assessment of the report and logbooks. This serves as a private prior that remains untouched throughout the debate and is only incorporated later during the final scoring. In parallel a *critique phase* takes place, where the evaluator launches a structured debate between a `Pro` and a `Con` agent. These agents argue for high and low scores, respectively, grounding their positions in concrete examples from the report itself and the underlying logbooks. The debate unfolds over three rounds, with each agent having access to the accumulated transcript, allowing them to respond to and challenge earlier points. After each debate iteration, a separate `Judge` model, unaware of prior rounds or previous scores, provides an explanation and raw score based solely on the arguments of the current debate. Finally, all judge explanations, scores, and private priors are aggregated and passed to a final reflective judge for

synthesis and scoring.

While the judge from each debate iteration provide meaningful justifications and raw scores, directly relying on a single integer value from a fixed Likert scale (e.g., 1 to 7) introduces its own limitations. Recent studies have highlighted a concerning lack of score variance when language models are asked to rate or classify responses. Park et al. [62] demonstrate that even when applying input-level perturbations such as prompt reordering, models such as GPT-3.5 tend to produce highly consistent outputs, typically centered around the middle of the scale. This central-tendency bias, where models default to "safe" middle values, results in low-variance distributions. Similar concerns are echoed in the work by Saynova et al. [63], who show that low-variance responses in LLM-generated samples can lead to inflated effect sizes and poor replicability detection. Their findings suggest that relying solely on deterministic outputs from a single pass fails to capture the uncertainty inherent in model reasoning, particularly in tasks involving subjective judgment.

To address this issue and better capture variability in model confidence, the evaluation design incorporates both structured prompt engineering and distributional scoring. During both the ideation and debate phases, all prompts are carefully constructed with explicit task descriptions, scoring criteria, penalization rules, and, in some cases, few-shot examples to guide the evaluator in assessing high- and low-quality responses. The prompts closely follow the structure of human evaluation guidelines and are designed to encourage confident, calibrated reasoning from the model, in line with the CoT-style prompting used in G-EVAL [11] (see Appendix C for example of evaluation instructions and structure). Additionally, during *Final Judge Scoring*, each final score is derived from a distribution over possible outputs rather than a single deterministic prediction. This is implemented via a sampling approach across the following temperature settings: 0.5, 1, 1.5. Unlike prior work that fixes temperature, it is treated here as a tunable parameter controlling output variance: low temperatures suppress variability, while high temperatures may introduce incoherence. The idea of sampling across this range is to surface not only what the model "typically" outputs, but also what it considers plausible under slight perturbations to the decoding process. This provides a more faithful approximation of the model's internal uncertainty and offers insight into how confident it is in its own estimates.

Each sample provides an integer score between 1 and 7, and the resulting empirical distribution is aggregated into probabilities $p(s_i)$ over each score value. The final score is then computed as the expected value:

$$\text{score} \;=\; \sum_{i=1}^{7} p(s_i)\, s_i \,,$$

where $p(s_i)$ is the normalized frequency of score s_i across the sampled completions. This formulation mirrors the G-EVAL approach [11], which originally used token-level log-probabilities to construct continuous evaluation scores. However, since access to log-probabilities is no longer available in current GPT-based APIs, this sampling-based method provides a practical alternative for approximating distributional confidence.

By generating a distribution instead of a single score, this method allows for more nuanced comparisons between reports, captures uncertainty in model judgment, and avoids scoring ties. While it is worth noting that the *human evaluation* maps one judge to one person, the Final Judge in this setup is not specifically intended to simulate an individual evaluator. It acts as a panel of evaluators that synthesizes multiple perspectives, including the Pro and Con agents across debate rounds, the intermediate judge scores, and the ideation prior, into a single calibrated judgment. Instead of mimicking a human, the idea is to consolidate the full multi-agent evaluation into a coherent probabilistic score that better reflects both uncertainty and argumentative strength, rather than using naive averaging across rounds.

3.6.3 Inter-Evaluator Agreement

Regardless of whether the evaluation is performed by humans or through automated LLM-based frameworks, inter-evaluator agreement remains a critical concept in assessing subjective tasks such as the quality of generated reports. This metric refers to the degree of consistency or agreement among different evaluators when assessing the same set of outputs. High inter-evaluator agreement is crucial because it indicates that the evaluation criteria are clear and the evaluators are reliably interpreting those criteria in the same way [49].

Text quality, particularly when generated by an automated system, is naturally subjective. Different evaluators may have different interpretations of what constitutes "good" or "relevant" text, based on their individual preferences, experiences, or expectations. As noted by Celikyilmaz et al., variability in evaluators' judgments can lead to inconsistent or biased results, which undermines the reliability of the evaluation process [49].

Taking this into consideration, both the inter-evaluator agreement of the automatic evaluation and the human evaluation will be calculated and compared. This will be done by calculating the variance in the evaluation scores, which allows for a better understanding of how consistent and in agreement the evaluators are.

3.6.4 Prompting Technique Evaluation

In addition to evaluating overall system performance, an ablation study is conducted at the component level to isolate the `Section Writer` and assess how different prompting techniques individually contribute to output quality and hallucination reduction. This evaluation is focused on one singular section written by the section writer utilizing the `Llama-33-70b` model. The section is generated using a constant input while varying only the prompt. This approach addresses a key limitation encountered when evaluating entire reports, namely the difficulty of tracing the origin of information. By restricting the writer to a single section with a fixed input, the aim of this setup is to provide the section writer with a known 'ground truth'. This allows for a more accurate assessment of how different prompts impact the output and to what extent they help mitigate hallucinations.

3.6.4.1 Prompt Evaluation Specifications

All prompt evaluations are based on a fixed input, primarily centered around the section header *"Fuel Cell System Issues"*. In addition to this, the section writer is provided with error, solution and image artifacts relevant to the header, the acronym list, and a pre-generated text of the previously written sections. This structure mirrors the input used by the actual writing agent, but with a known and constant input between the different prompts.

Furthermore, when it comes to the structure of the prompts themselves, the differences between the prompts are described below:

- **No Chain-of-Thought:** For this prompt the additional instructions on how to separate the task into different steps are entirely removed, while keeping the same information in the input. The exact differences are displayed in box 1 in Figure 3.6 below.
- **No One-shot prompting:** For the prompt without one-shot prompting the only difference is that the additional context is removed from the prompt. This variable provides the section writer both with an example for the structure of the artifacts as well as examples and guidelines on how to properly utilize these for citations. The exact difference can be seen in box 2 in Figure 3.6 below.
- **No RAG Partitioning:** In this version, the prompt remains unchanged, but the "Data" artifact is modified to include all artifacts from the entire analysis, rather than restricting them to those relevant only to the current header. This means the setup still relies on RAG, but without the refined, title-specific retrieval. The exact change is shown in box 3 in Figure 3.6 below.

```
example_prompt = (
    f"### Section Writer Request\n"

    f"### INSTRUCTIONS:\n"
    f"You are a scientific expert report writer responsible for composing individual sections of a comprehensive report that is going to be published at a conference.\n"

1.  f"Please ensure you fully understand the concepts and the technical abbreviations provided below before proceeding. Think step by step.\n\n"
    f"Your task is to write the section for the given **Header:**."
    f"To do so, use the provided **Data:** as a guiding resource, and carefully review the following **Abbreviations and Explanations** to understand all technical terms in context.\n"
    f"Also, take into account any **Prior Sections:** for consistency and to avoid redundancy. Use **Used Abbreviations** to check if an abbreviation has been already been defined and explained.\n"

2.  f"**Additional Context:** \n\"\"\"{prompt_context}\"\"\"\n\n"

    f"### IMPORTANT:\n"
    f"- Only produce a section if the provided content contains sufficient **Data:**.\n"
    f"- Do not invent or hallucinate any information.\n"
    f"- If insufficient **Data:** is available, output ONLY and EXACTLY: 'Not sufficient data for producing this section'.\n"
    f"- Do not repeat the header (avoid starting with the header text or markdown headings).\n"
    f"- Use succinct, clear, and formal language, ensuring every word is purposeful.\n"
    f"- STRICTLY adhere to **Used Abbreviations**: if an abbreviation has been defined earlier, do not include its explanation again; only the short form must appear in all subsequent sections.\n\n"
    f"======================\n\n"
    f"**Header:** \n\"\"\"{header}\"\"\" \n\n"
    f"**Abbreviations and Explanations:**\n\"\"\"{acronyms}\"\"\"\n\n"
    f"**Used Abbreviations**:\n\"\"\"{used_abbreviations}\"\"\"\n\n"

3.  f"**Data:** \n\"\"\"{artifacts}\"\"\"\n\n"

    f"**Prior Sections:**\n\"\"\"{previous_section}\"\"\"\n\n"
    f"### Section Text:\n"
)
```

Figure 3.6: The differences between the prompts used in the prompt evaluation pipeline. Box 1 is removed for no Chain-of-Thought, box 2 is removed for no One-shot prompting and box 3 is changed from "header_specific_artifacts" to "artifacts" when changing the RAG.

3.6.4.2 RAGAS Scoring Framework

The actual score used in the prompt evaluation pipeline is based on the RAGAS framework, or specifically the automatic metrics for *Faithfulness*, *Factual Correctness* and *Summary Score* provided by RAGAS [64]. These metrics assess the factuality of the written section in three different ways:

- **Faithfulness:** The *Faithfulness* measures how well the generated response aligns with the retrieved context, where low scores typically indicate hallucinations caused by introducing unsupported or fabricated facts not present in the artifacts.
- **Factual Correctness:** *Factual Correctness* evaluates the factual accuracy of information based on a reference text, which in this case is the same *"Fuel Cell System Issues"* section but from the full report. While this reference in itself is not guaranteed to be completely error-free, it complements the other metrics by providing an additional independent layer of verification.
- **Summary Score:** *Summary Score*, although not a direct measure of hallucinations, instead checks whether the generated section can effectively utilize the content of the provided artifacts. It focuses less on factual accuracy and more on whether the model can actually utilize and effectively present the information provided in the artifacts in an understandable manner.

Utilizing these metrics, the RAGAS evaluation framework offer a structured and multifaceted approach to systematically detect and quantify hallucinations, with each metric capturing a distinct and independent aspect of output reliability. When combined, their complementary nature offers a more balanced foundation for evaluation, making it easier to draw clear and unbiased conclusions about the effectiveness of the applied techniques in reducing hallucinations.

4

Results

This chapter presents the results of the experiments conducted in the study. The findings are divided into two main parts. The first part evaluates the performance of the reports generated by the MAS in comparison to both a human-written report and a single LLM Baseline. These results are displayed for both the human evaluators and the LLM based evaluation system. The evaluation includes average scores and variance across predefined metrics, complemented by qualitative insights and comments on the reports. The second part focuses on assessing the performance of the different prompting techniques using automatic RAGAS evaluation metrics.

4.1 Human Evaluation Results

This section presents the results from the survey. The numerical results are presented in Figure 4.1, which displays the average score for each of the metrics between the reports. The variance in score for the different reports are displayed as error bars for each of the metrics.

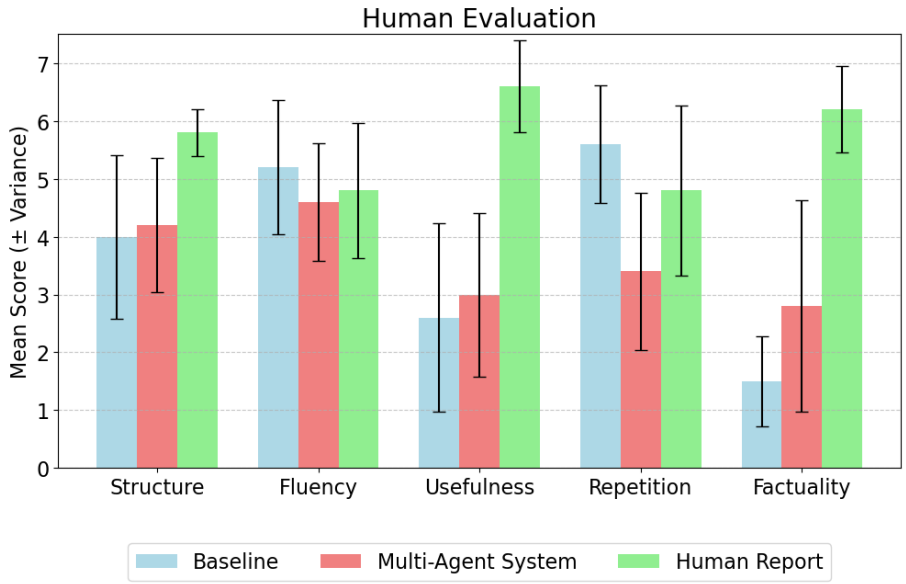

Figure 4.1: Average score and variance for the 5 metrics from the human evaluators in the survey. A higher score indicates better performance.

As can be seen in Figure 4.1 the Baseline and the MAS performs relatively similar when it comes to *Structure* and *Fluency*. For *Repetition*, the Baseline performs best, closely followed by the human report, both with a notably higher score than the MAS. Lastly, the manually written report significantly outperforms both the Baseline and agent based system in both *Usefulness* and *Factuality*. However, the MAS is still better than the Baseline when assessing *Factuality*.

Assessing the aggregated results across all metrics, displayed in Table 4.1 & 4.2 below, the Human Report significantly outperforms the MAS and the Baseline, with an average score of 5.64 compared to 3.60 and 3.78. In addition to this, the variance is the lowest for the human report ($\pm$0.98), followed by the Baseline at ($\pm$1.53) and lastly the MAS at ($\pm$1.92). Assessing the variance of the individual metrics instead, the *Usefulness*, *Repetition* and *Factuality* have the highest variance with $\pm$1.76, $\pm$1.68, and $\pm$1.51 respectively.

Table 4.1: Mean Scores across metrics and systems for the Human evaluation.

Metric	**Baseline**	**Multi-Agent System**	**Human Report**	**Overall Avg**
Structure	4.00	4.20	**5.80**	4.67
Fluency	**5.20**	4.60	4.80	**4.87**
Usefulness	2.60	3.00	**6.60**	4.07
Repetition	**5.60**	3.40	4.80	4.60
Factuality	1.50	2.80	**6.20**	3.50
Average	3.78	3.60	**5.64**	**4.34**

Table 4.2: Variance across metrics and systems for the Human evaluation.

Metric	**Baseline**	**Multi-Agent System**	**Human Report**	**Overall Avg**
Structure	**2.00**	1.36	0.16	1.17
Fluency	**1.36**	1.04	**1.36**	1.25
Usefulness	**2.64**	2.00	0.64	**1.76**
Repetition	1.04	1.84	**2.16**	1.68
Factuality	0.60	**3.36**	0.56	1.51
Average	1.53	**1.92**	0.98	**1.47**

The scoring trends are further reflected in the qualitative responses section of the survey, where evaluators were asked to justify their scores and provide examples. As shown in Table 4.3, some selected quotes from the evaluators are displayed to offer some additional insights into their assessments.

Table 4.3: Selected Qualitative Feedback from Human Evaluators

Title	**Experience with Engineering Reports**
Specialist ESW Application Engineer	**Written Several ERs**
"The report generator is quite good at collecting information and at gathering them in categories. It is also quite good at setting a structure and writing text even though some sentences don't make sense or include heavy mistakes. In terms of value of the report generated, I would say the report does not fulfill the purpose of describing the test results and analysis and summarizing the outcome, it focuses on issues that happened during the expedition and ways to solve them or mitigate them. The problem resides mostly in the content of the logbook which focuses almost exclusively on issues and mitigations solutions instead of analysis of the tests which is usually done after the test/expedition".	
"The tool has potential but needs to be improved by providing it with more relevant data but also by including some technical knowledge such as units."	
Data Scientist	**Read some ERs**
"The main disadvantage was that it did not understand the images. Also felt like it was a bit all over the place. More like reading a summary of the tests, but not in the order they happened. Also, for me that was not there for the test, I felt that the agent was missing the deeper understanding and the analysis produced could mostly be removed. However, a VERY good template to start from!"	
Test Engineer	**Written Several ERs**
"The advantages of report are it described all the technical problems we faced on the expedition."	
Expert Propulsion System V&V Engineer	**Written Several ERs**
"The conclusions in the report were too generic to add to technical knowledge for the reader. The overall language of the report is more of a management summary than a technical report describing test cases, analyses, and conclusions."	

The quotes in Table 4.3 provide qualitative feedback from the evaluators and highlight several interesting points. Many evaluators noted that the report generator was effective at collecting information, organizing the content, and creating a clear structure and writing flow. In addition to this, there was a general agreement that

the generated reports focused mainly on the technical issues encountered during the expedition, while giving less attention to test results, outcomes, or post-test analysis. Some evaluators also pointed out that the reports lacked enough technical analysis and did not offer deeper insights or interpretations of the test data. Lastly, concerns were raised about the system's inability to interpret images, which could result in the loss of useful information. However, despite these issues, the evaluators agreed that the report served as a good starting point or template for drafting a report manually.

4.2 LLM Evaluator Results

This section displays the results from the LLM based evaluator on the same three reports as for the human evaluation. The results follow a similar structure, with average scores and variance for each of the five metrics together with a short motivation. The results from the evaluation are displayed in Figure 4.2 below.

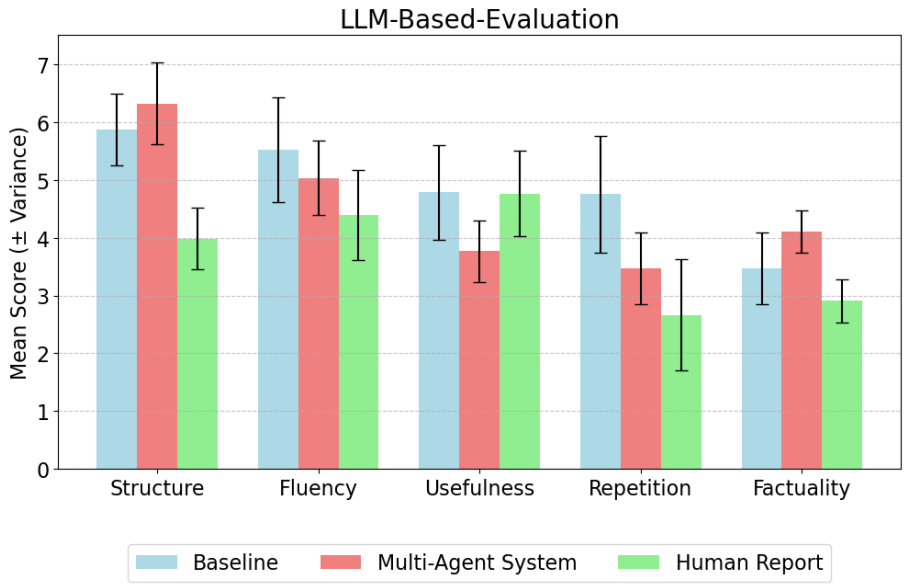

Figure 4.2: Average score and variance for the 5 metrics from the LLM based evaluator.

As can be seen in Figure 4.2, the results from the LLM based evaluator is notably different to the human evaluation, both in mean and variance. Addressing the aggregated results across the models, it is evident that the LLM based evaluator provides somewhat higher scores with an average of 4.52 compared to 4.36 for the human evaluators. Assessing this further, for each of the reports individually, the reports generated by the Baseline and MAS are both scored significantly higher by the LLM based evaluator (4.88 compared to 3.78) and (4.54 compared to 3.60), whereas the manually written report instead received a lower score on average (3.74 compared to 5.64) when compared to the human evaluators. In addition to this the variance of the LLM based evaluation is significantly lower across all reports, with a notable reduction in average variance (1.33 to 0.52).

Table 4.4: Mean Scores across metrics and systems for the LLM based evaluation system.

Metric	Baseline	Multi-Agent System	Human Report	Overall Avg
Structure	5.87	**6.32**	3.98	**5.39**
Fluency	**5.52**	5.03	4.39	4.98
Usefulness	**4.78**	3.77	4.76	4.44
Repetition	**4.75**	3.47	2.67	3.63
Factuality	3.47	**4.11**	2.91	3.49
Average	4.88	4.54	3.74	**4.39**

Table 4.5: Variance across metrics and systems for the LLM based evaluation system.

Metric	Baseline	Multi-Agent System	Human Report	Overall Avg
Structure	0.38	**0.78**	0.28	0.48
Fluency	**0.82**	0.41	0.61	0.61
Usefulness	**0.67**	0.28	0.55	0.50
Repetition	1.02	0.38	**0.92**	**0.78**
Factuality	**0.39**	0.14	0.14	0.22
Average	**0.66**	0.40	0.50	**0.52**

4.3 Correlation Score between Evaluations

Table 4.6 presents the Pearson correlation coefficients, which essentially measures how the variables move together by dividing their covariance by the product of their standard deviations, between human evaluation scores and LLM-based evaluation scores for each metric. The correlation scores vary considerably across the five evaluated aspects. *Structure* shows the strongest negative correlation at -0.96. Similarly *Factuality* also demonstrates a negative correlation of –0.67. In contrast, *Fluency* shows the highest positive correlation (0.59), followed by *Repetition* (0.48) and *Usefulness* (0.40). These values reflect how similarly or differently each evaluation method scores the same systems, with higher absolute values indicating stronger linear relationships, positive or negative, between the human and LLM-based assessments.

Table 4.6: Pearson correlation between Human Evaluation and LLM-Based Evaluation across metrics.

Metric	Pearson Correlation (r)
Structure	-0.96
Fluency	0.59
Usefulness	0.40
Repetition	0.48
Factuality	-0.67

The presence of strong negative correlations for Structure and Factuality suggests fundamental differences between how human evaluators and LLM-based evaluation interpret these metrics, raising important questions about the suitability and

interpretability of LLM-based scoring for these specific dimensions.

4.4 Ablation study

Lastly, this section displays the results from the ablation study of the prompts for a singular section, namely an analysis section about *Fuel Cell System Issues*. The scoring metrics from RAGAS outputs a direct score for each of the reports. These scores are presented in Table 4.7 below.

Table 4.7: Average evaluation scores of the RAGAS metrics per prompt type across 5 rounds of generated sections.

Prompt Type	**Faithfulness**	**Factual Correctness**	**Summary Score**
Full Prompt	0.9136	0.6480	0.5284
No CoT	0.8926	0.6080	0.5053
No One-Shot	0.8752	0.4680	0.3831
No RAG	0.3455	0.0840	0.2219

As can be seen in the table, the prompt containing all techniques performs the best across all metrics. This is closely followed by the prompt without CoT and then by the prompt with no One-shot examples. The prompt without One-shot performs relatively similar to the full prompt on *Faithfulness*, but has a notable drop in both *Factual Correctness* and *Summary Score*. The prompt that performs worst is the one without the enhanced RAG artifacts, which has a significantly lower score than the other prompts across all metrics. Most notably for the *Factual Correctness* and *Faithfulness*, for which the decline is most distinct.

While the previous LLM-based evaluation revealed some notable differences compared to human assessments displayed in Section 4.3, RAGAS metrics are applied within the limited context of a single section rather than the full report. As such, while assessing an inherently subjective metric such as usefulness of an entire report can be a rather difficult task, the constrained context of the single section, together with the constant and well defined input provides a stronger foundation for evaluating factuality through RAGAS. The main differences compared to the other LLM-based evaluator, is firstly the presence of a ground truth, in the form of the known input, and secondly that the rather subjective grading task is transformed into a more well defined comparison task between a source (i.e the artifacts) and a summary (i.e the section).

5

Discussion

This section reflects on the key findings from both the LLM-based evaluation and the human survey-based assessment, but also using the automated RAGAS metrics. The primary focus of the discussion revolves around analyzing the performance of the MAS, but it also address the quality of the evaluation itself and how the results should be interpreted. Furthermore, the results revealed that the MAS has its strengths in structuring and summarizing information, while lacking in deeper analytic aspects. The chapter is finalized by stating some potential future work, together with a conclusion.

5.1 System Performance

The human evaluation results reveal several clear trends in how the different systems were perceived across key metrics. As seen in Table 4.1 and Figure 4.1, the manually written human report consistently outperformed both the baseline and the MAS across nearly all metrics in the human evaluation. Its overall average score (5.70) was substantially higher than that of the MAS (3.60) and the baseline (3.78).

Despite having a lower score on average, the MAS demonstrated several notable strengths based on the feedback from evaluators. Reviewers highlighted its ability to organize content into meaningful categories and maintain a coherent structure also reflected in the high scores for the *Fluency* and *Structure* metrics. For example, the `Specialist ESW Application Engineer` noted that *"The report generator is quite good at collecting information and at gathering them in categories..."*, and that it *"is also quite good at setting a structure and writing text..."*, even if the content sometimes lacked depth or precision. Others appreciated the clarity and grammatical correctness of the writing, describing the text as grammatically correct and easy to read, and recognized its potential as a solid starting point for more refined reporting.

Additionally, as mentioned by the `Test Engineer`, the system successfully identified and described many of the technical issues from the expedition, by referencing directly to the original error entry in the logbook. Several evaluators found this helpful, as it provided a solid structure and summary of relevant and interesting events that engineers could enhance with more detailed insights. This is supported by the comment from the `Data Scientist`, who described the report as a *"VERY good template to start from,"* suggesting that the MAS could serve as a valuable tool to build upon, when combined with human domain expertise.

Although the system could generate structurally coherent and fluent text, it was notably lacking in both *Usefulness* and *Factuality*. However, while the MAS was outperformed by the human report, it did still achieve a significantly higher factuality score than the baseline, indicating that its architecture and prompting strategy did in fact help reduce factual errors.

5.2 System Limitations

The qualitative feedback from human evaluators, visualized in Table 4.3, reinforces many of the trends seen in the quantitative results, while also providing a deeper insight into the current system's limitations and potential. One of the most consistent criticisms was centered on the system's lack of specialized domain knowledge. Particularly in technical fields, producing truly "useful" or "factual" content requires a level of expertise that the LLMs, even when enhanced with retrieval and multi-agent structuring, struggle to understand fully in this case. The model only had access to the expedition logs, which themselves mainly reported issues. Consequently, it naturally generated a report that focused heavily on problems and solution strategies. As mentioned by the `Expert Propulsion System V&V Engineer`, *"The overall language of the report is more of a management summary than a technical report describing test cases, analyses, and conclusions."*, thus lacking the depth and critical analysis typically expected from such documents. This can be related to limitations not of the model itself, but rather the provided artifacts or data. Without access to test specifications, signal data, or post-test engineering evaluations, the system is naturally unable to generate meaningful insights or conclusions beyond the surface-level observations already present in the logbooks.

Additionally, a large portion of advanced technical content, such as signal data connected to the faults, was conveyed in a graphical plot format, making it impossible for a text-only LLM system to interpret them meaningfully. Even when paired with a vision model, simply "seeing" the plot is not sufficient; the system lacks the engineering reasoning capabilities needed to identify trends, anomalies, or causal relationships within the data and instead only states values without understanding the underlying meaning of the specific segment of data displayed in the plot. This gap was highlighted by the `Data Scientist`, which describes that "*The main disadvantage was that it did not understand the images*".

5.3 Influence of Prompting Techniques

Analyzing the results from the prompt evaluation, illustrated in Table 4.7, the ablation study showed a clear downward trend in performance as individual prompting techniques were removed. The Full Prompt configuration, combined with CoT reasoning, One-Shot examples, and partitioned RAG-based retrieval, consistently achieved the highest scores across all three RAGAS metrics; *Faithfulness*, *Factual Correctness*, and *Summary Score*. Notably this evaluation pipeline was not compared to a human gold-standard, but was instead setup under more distinct conditions to

reduce the uncertainty of LLM based evaluation. Instead of letting the LLM score the generated section directly, the input and section are both divided into individual claims by the RAGAS metrics, which was then used to calculate scores according to predefined mathematical formulas. Thus, while the evaluation process still contains some variability, it employs a more deterministic approach which should be efficient and objective in capturing the factuality metric of the generated sections.

Among the tested strategies, enhanced RAG emerged as the most effective in preventing hallucinations, as seen in the substantial drop in factuality scores when it was excluded. Notably, the *Summary Score* and *Faithfulness* dropped considerably much, indicating that the generated section is both clearly different from the original reference section, and make a lot of statements that cannot be grounded in the artifacts partitioned for this specific section. This highlights a key limitation of the system; even when provided with clear instructions, such as focusing specifically on Fuel Cell System Issues in this case, it struggles to independently determine which parts of the input are relevant. Without structured guidance, the model may include unrelated or repetitive information. By preprocessing and partitioning the artifacts, the system is provided with distinct information and was thus better on targeting contextually appropriate details, which led to significantly improved correctness and overall better output quality.

While One-Shot examples and CoT reasoning also contributed to improving the factuality, their individual impact was smaller compared to the effect of RAG. Removing CoT resulted in only a slight drop across all metrics, suggesting only minor influence, possibly due to the smaller parameter size of the Llama-33-70b model used for the section generation of this separate evaluation pipeline. However, while removing One-Shot prompting had little effect on faithfulness, it caused a noticeable decline in both *Factual Correctness* and *Summary Score*. This suggests that the content remained grounded in the partitioned artifacts, but the quality and richness of how the information was presented suffered without the example guidance, which is reflected in the lower scores.

Still, each individual technique did improve the factuality across all metrics. One interesting aspect is also that each technique address a different aspect of the generation process. Partitioned RAG enhanced artifact relevance, One-Shot prompting helped define expected output structure, whereas CoT supports logical reasoning. Considering their individual contribution to factual output, their combined effects could all help reduce the overall risk of hallucination due to their complementary nature of addressing sources of hallucinations in different points of the generation pipeline.

5.4 Assessment of Evaluation Frameworks

A comparison of overall average scores across the evaluation systems reveals a clear asymmetry in preference between human and LLM-based assessments. As shown in Table 4.1, human evaluators gave the highest overall score to the human-written report (5.64), noticeably above both the MAS (3.60) and the Baseline (3.78). In

contrast, the LLM-based evaluation, shown in Table 4.4, exhibits the opposite trend: the Baseline and MAS reports received higher average scores (4.88 and 4.54 respectively), while the human report was rated lowest (3.74). This reversal reflects a clear bias in both directions, with human evaluators favoring the human report and the LLM-based evaluation favoring LLM-generated outputs.

This divergence is further explained by the correlation scores presented in Table 4.6. Metrics such as Structure ($r = -0.96$) and Factuality ($r = -0.67$) show strong negative correlations, suggesting that human raters perception of *"structurally sound"* or *"factually accurate"* is often not reflected in the LLM's assessments. Structure, in particular, illustrates a clear bias in the LLM-based evaluation. LLM-generated reports receive the highest scores in this category (6.32 for the MAS System and 5.87 for the baseline), while the human-written report receives the lowest (3.98), as shown in Table 4.4. This suggests that the LLM favors reports that mirror its own stylistic and structural conventions, likely valuing coherence patterns that align with its training data.

Factuality, however, reveals a more complex and fundamental difference in how human and LLM-based evaluations interpret accuracy. In the LLM-based evaluation (Table 4.4), both LLM-generated reports receive higher factuality scores than the human-written report. This can be attributed to the fact that the LLM determines factual accuracy by directly comparing the report content to the logbooks, which serve as the structured input data. Since the generated reports were conditioned on this data, they are more likely to contain statements that can be verified. In contrast, the human-written report is not entirely based on the logbooks and often utilize prior domain expertise or external knowledge that is not present in the logbook data. As a result, it receives a lower score when judged by the LLM because it cannot validate content beyond the information it has access to. However, the opposite pattern is observed in the human evaluation, displayed in Table 4.1, where the human-written report receives the highest factuality score (6.20). This suggests that human evaluators do not rely solely on the logbooks but instead assess accuracy using their own background knowledge and professional experience. As a result, they are better equipped to recognize valid external information that the LLM cannot verify. Interestingly, regardless of the discrepancies between the evaluation frameworks, the MAS shows a consistent trend toward higher factuality scores compared to the baseline. While this difference is not statistically significant, it may suggest that the MAS produces outputs more closely aligned with logbook content and potentially less prone to hallucination, whether factuality is judged against structured data or based on expert-level understanding.

Conversely, Fluency ($r = 0.59$) and Repetition ($r = 0.48$) show more moderate positive correlations, indicating that both evaluation frameworks converge on these dimensions. This alignment likely stems from the fact that these metrics rely on observable linguistic features such as grammar, syntax, and lexical repetition. Because the criteria are more concrete and easier to define, both LLMs and humans can evaluate them with greater consistency and less reliance on subjective interpretation or external knowledge.

Surprisingly, Usefulness also shows a positive correlation ($r = 0.40$), despite being the most abstract and subjective of all metrics. Unlike fluency or repetition, usefulness is inherently more difficult to define, even for human evaluators, as it depends on individual expectations, context, and the perceived purpose of the report. This complexity is even greater for an LLM, which does not have access to the intended use case of the report and therefore lacks a clear basis for judging how well the content serves its function. In the human evaluation, displayed in Table 4.1, usefulness scores vary significantly across systems, ranging from 2.60 for the baseline to 6.60 for the human report, reflecting evaluators' strong preferences. In contrast, the LLM-based scores in Table 4.4 remain clustered within a narrower range, with all systems scoring between 3.77 and 4.78. This limited variation suggests that the LLM has difficulty distinguishing between degrees of usefulness and may tend to default toward middle-range scores. The resulting correlation is therefore possibly a consequence of score compression rather than a meaningful agreement in evaluative judgment.

In conclusion, the weak correlations between the two evaluation frameworks highlight the automated LLM-based system's limited capacity to recognize contextually relevant or externally validated information. While internally consistent, the automated evaluation does not fully align with human evaluators, particularly for metrics that require nuanced interpretation or external domain expertise. Despite these limitations, the LLM-based evaluation remains valuable due to its consistency, scalability, and objective grounding in structured input data, especially for metrics clearly defined by observable linguistic features such as Fluency and Repetition. Ultimately, however, these findings emphasize that while the automated evaluation can effectively complement human judgment, caution should be exercised when using it independently for assessing metrics that heavily depend on external knowledge, subjective interpretation, or nuanced reasoning.

5.4.1 Inter evaluator agreement

As shown in Table 4.2 and Table 4.5, there is a notable difference in score variance between the human evaluation and the automated LLM-based system. Variance is consistently higher across all metrics in the human evaluation, indicating lower agreement among evaluators. This variability highlights the difficulty of consistently assessing qualitative dimensions of language generation, especially when evaluators bring different expectations, interpretations, or prior knowledge to the task. In contrast, the automated system exhibits lower variance, suggesting greater internal consistency and confidence in its assessments.

However, this difference can potentially be deduced back to structural differences in how the evaluations are conducted. In the human evaluation, scores are aggregated across multiple annotators, each with their own interpretations and thresholds, which naturally introduces interpersonal variation. Conversely, the automated system relies on a single reflective `Final Judge` that produces one consolidated score derived from multiple generations under varied sampling conditions. In this context, variance in the automated setup reflects the model's internal uncertainty across

decoding perturbations, not disagreement between individuals. Therefore, variance comparisons across the two systems are not directly equivalent but still provide meaningful insight when interpreted within their respective frameworks.

5.4.1.1 Variance Patterns in Automated Evaluation

Variance scores from the automated LLM-based evaluation (Table 4.5) reveal clear differences in reliability across the Baseline (0.66), Multi-Agent System (0.40), and Human-written (0.50) reports. The Multi-Agent System report consistently achieves lower variance across nearly all metrics, especially pronounced in Repetition (0.38), Usefulness (0.28), and Fluency (0.41). This suggests that the automated evaluator found the Multi-Agent-generated report relatively straightforward and unambiguous to assess, implying that the content provided fewer internal conflicts or uncertainty during multi-round argumentative critique. In contrast, the Baseline report exhibits notably higher variances, particularly in Fluency (0.82) and Repetition (1.02), indicating that the automated evaluation experienced significant internal uncertainty or conflicting assessments in these dimensions across the decoding perturbations.

The Human-written report demonstrates an intermediate pattern of overall variance (0.50) but interestingly achieves the lowest variance for Factuality (0.14), identical to the Multi-Agent System. This low variance can be attributed to the evaluation framework's design, where factual accuracy is assessed explicitly by comparing report content against structured logbooks. Since both the Multi-Agent System and Human-written reports either align precisely or consistently diverge from the logbook data, the evaluator can confidently assign scores, high for logbook-aligned content and consistently lower for information that falls outside this structured input. Thus, the evaluator's certainty stems directly from the presence or absence of verifiable logbook evidence.

Ultimately, these contrasts serve as a diagnostic: high variance pinpoints precisely what metrics the evaluation guidelines or prompt examples could benefit from greater clarity and refinement, while low variance highlights what metrics in which the automated evaluator already achieves definitive, consensus-like judgments.

5.4.1.2 Variance Patterns in Human Evaluation

Several metrics in the human evaluation display particularly high variance across systems, revealing how challenging certain aspects are to evaluate consistently. For instance, Usefulness shows a variance of 2.64 for the baseline and 2.00 for the MAS, compared to just 0.64 for the human report. Repetition and Factuality also show notable variation. The human report has the highest variance in repetition (2.16), indicating disagreement on how much repetition is acceptable or even noticeable in that context. Similarly, the MAS shows the highest variance in factuality (3.36), suggesting that annotators had differing views on the correctness of its content, possibly due to varying familiarity with the underlying technical material.

This variability highlights the difficulty of consistently assessing qualitative dimensions

of language generation, especially in technical contexts where understanding may depend heavily on prior expertise. Even when evaluators are given clear definitions, their interpretations of what constitutes a "useful" insight, "factual" statement, or a "well-structured" report can differ significantly. Furthermore, different types of reports may achieve quality in different ways: a short, concise report may appear well-structured due to its simplicity, while a longer one may be structurally sound through complexity and detail. These differences make it difficult to apply evaluation criteria uniformly and reinforce the inherently subjective nature of human judgment. Consequently, while human evaluations offer both critical depth and contextual awareness, they also introduce greater variability, which is reflected in the higher variance values across several metrics.

5.5 Future work

Taking these aspects, discussed in the chapter, into consideration. One of the key limitations identified in the study is the system's constrained ability to perform deep analysis and reasoning about technical content, despite showing clear promise in structuring and summarizing information. Considering that the LLMs are originally trained on a generic corpus, rather than Volvo specific information, there is essentially no way for the model to achieve any deeper understanding than what is presented from the specialized data provided in the in-context learning. Thus, this shortcoming becomes closely linked to the limited quality and contextual depth of the input data. Currently, the MAS relies solely on expedition logbooks, which are primarily issue-focused and often lack advanced contextual information regarding the test.

With this in mind, future work should prioritize deepening the understanding of agents within the system. In the current setup, agents are designed to divide tasks such as information gathering, writing, or summarizing. However, they all essentially still function as generalists, since they operate on the same limited information derived solely from the logbooks. This uniform and narrow input scope restricts each agent's ability to perform nuanced, domain-specific reasoning. Moving forward, a more effective system could involve agents fine-tuned on distinct domains of knowledge, supported by richer and more diverse datasets. For example, one agent could specialize in truck architecture and component behavior, another in the underlying physics, and a third in qualitatively interpreting sensor data. By equipping agents with targeted expertise and relevant training data, the agents of the system could more efficiently complement each other by dividing not only the task itself but also their specialized knowledge required in the analysis. Notably, this approach was not current feasible, due to the lack of structured and domain-specific training data.

Another direction for future work involves re-imagining the role of the system, not as a fully autonomous report generator, but rather as a template generation tool that is complemented with human domain expertise. In its current form, the system performs well in organizing known issues, referencing relevant events, and producing readable and well-structured output. However, it lacks the ability to validate technical facts or derive deeper engineering insights reliably. To capitalize on these structural strengths and mitigate the analytical limitations, the system could be integrated with

a human-in-the-loop workflow. In this setup, domain experts would provide brief annotations, analyze key findings, or offer extended input, essentially working as a domain expert node providing advanced insights to the current system in a RAG-like manner. The system could then structure, elaborate on, and polish this extended analytical input into a more technically advanced and useful draft. Alternatively, it could serve as a pure template generator, providing an aggregated starting point that engineers can further refine.

5.6 Conclusion

To summarize, this thesis set out to investigate whether a MAS could outperform a manually written report in the context of technical report generation. While the initial goal was to assess its performance, the project evolved into a broader exploration of how quality in generated reports can be meaningfully assessed. An endeavor that proved to be as challenging as the generation task itself.

The MAS, while ultimately outperformed by the human report in most human evaluation metrics, demonstrated several promising capabilities. Notably, it produced fluent, well-structured text and showed clear potential in identifying and categorizing key technical events. These strengths suggest that the MAS architecture can serve as a powerful scaffolding tool, offering a coherent foundation that experts can build upon. In addition to this, the notable improvement in factuality compared to the baseline system further highlight the effectiveness of the architectural choices and prompting techniques, particularly the partitioned RAG.

However, the system's limitations, especially in factual depth, technical insight, and domain-specific reasoning, displays critical challenges in relying solely on language models for report generation tasks. The MAS struggled to replicate the nuanced understanding and context-awareness that human experts naturally bring, particularly when working with limited input data.

Equally important was the finding that evaluation itself is a nontrivial problem. Although human evaluation is often considered the gold standard for assessing language generation tasks, our results revealed notable variance in inter-evaluator agreement, even when detailed and standardized instructions were provided to minimize bias. This variability underscores the inherently subjective nature of language evaluation, particularly in report writing, where individual backgrounds, expectations, and interpretations can significantly influence judgments.

In addition to this, discrepancies were observed between human and LLM-based assessments, with each framework favoring outputs aligned with its own background. While LLM-based evaluation favored structurally coherent and artifact-grounded content, human evaluators prioritized domain relevance, external knowledge integration, and previous experience. Notably, the LLM-based evaluation showed stronger alignment with human judgments on metrics that are well-defined, less subjective, and grounded in observable linguistic patterns, suggesting that such aspects may be more reliably automated. Thus, although human evaluation should continue to be regarded as the gold standard, LLM-based evaluation can still serve as a practical and complementary tool, especially in large-scale and clearly defined assessment settings.

Ultimately, while the MAS did not surpass human-authored reports in overall quality, it represents a meaningful step toward integrating automation into report generation. Its ability to structure, summarize, and reference complex input data makes it a promising candidate for use as a template generator. Future systems could benefit from specialized fine-tuning and human-in-the-loop frameworks that combine LLM efficiency with expert human judgment.

Bibliography

[1] E. Dantas Costa et al. "Potential of artificial intelligence to generate health research reports of decayed, missed and restored teeth". In: *Odovtos International Journal of Dental Sciences* 26.2 (2024), pp. 14–19.

[2] V. Tiwari et al. "Automatic generation of chest x-ray medical imaging reports using lstm-cnn". In: *Proceedings of the International Conference on Data Science, Machine Learning and Artificial Intelligence*. 2021, pp. 80–85.

[3] Jiwoo Park et al. *Patient-centered Radiology Reports with Generative Artificial Intelligence: Adding Value to Radiology Reporting*. `https://www.nature.com/articles/s41598-024-63824-z`. Accessed on May 13, 2025. 2024.

[4] R. Gupta, G. Pandey, and S. K. Pal. "Automating Government Report Generation: A Generative AI Approach for Efficient Data Extraction, Analysis, and Visualization". In: *Digital Government: Research and Practice* (2024).

[5] D. S. Pinto et al. "Comparison of an AI-Generated Case Report With a Human-Written Case Report: Practical Considerations for AI-Assisted Medical Writing". In: *Cureus* 16.5 (2024).

[6] Mert Cemri et al. *Why Do Multi-Agent LLM Systems Fail?* `https://arxiv.org/html/2503.13657v1`. Accessed on May 13, 2025. 2025.

[7] L. Huang et al. "Addressing Hallucination in Large Language Models: Challenges and Strategies". In: *AI Research Journal* 27.4 (2023), pp. 567–589.

[8] Ismail Siddiqui. *RealityNet: A Multi-Agent Framework for Hallucination Detection in Large Language Models*. `https://mohdismailsiddiqui011.medium.com/realitynet-a-multi-agent-framework-for-hallucination-detection-in-large-language-models-8eedc57f18df`. Accessed on May 13, 2025. 2025.

[9] Luke Yoffe, Alfonso Amayuelas, and William Yang Wang. *DebUnc: Mitigating Hallucinations in Large Language Model Agent Communication with Uncertainty Estimations*. `https://arxiv.org/html/2407.06426v1`. Accessed on May 13, 2025. 2024.

[10] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. "Why we need new evaluation metrics for NLG". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017, pp. 2241–2252.

[11] Yang Liu et al. *G-EVAL: NLG Evaluation using GPT-4 with Better Human Alignment*. `https://arxiv.org/abs/2303.16634`. Accessed on April 29, 2025. 2023.

[12] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[13] Wenlong Ji et al. “An Overview of Large Language Models for Statisticians”. In: (2025). arXiv: 2502.17814 [stat.ML]. URL: https://arxiv.org/abs/2502.17814.

[14] Alec Radford et al. “Improving Language Understanding by Generative Pre-Training”. In: *OpenAI blog* (2018). URL: https://openai.com/research/language-unsupervised.

[15] Ari Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019, pp. 2341–2351. DOI: 10.18653/v1/P19-1237. URL: https://arxiv.org/abs/1904.09751.

[16] Jeremy Howard and Sebastian Ruder. “Universal Language Model Fine-tuning for Text Classification”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2018. URL: https://arxiv.org/abs/1801.06146.

[17] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *Journal of Machine Learning Research* 21 (2020), pp. 1–67. URL: http://jmlr.org/papers/volume21/20-074/20-074.pdf.

[18] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *arXiv preprint arXiv:2005.14165* (2020). URL: https://arxiv.org/abs/2005.14165.

[19] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. https://arxiv.org/abs/2001.08361. Accessed on May 13, 2025. 2020.

[20] Noam Shazeer et al. “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Accessed on May 13, 2025. 2017.

[21] Dzmitry Lepikhin et al. *GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding*. https://arxiv.org/abs/2006.16668. Accessed on May 13, 2025. 2020.

[22] Yi Tay et al. *Sparse Sinkhorn Attention*. https://arxiv.org/abs/2002.11296. Accessed on May 13, 2025. 2020.

[23] Hugo Touvron et al. *LLaMA 2: Open Foundation and Fine-Tuned Chat Models*. https://arxiv.org/abs/2307.09288. Accessed on May 13, 2025. 2023.

[24] DeepSeek AI. *DeepSeek R1: A 672B MoE Language Model*. https://arxiv.org/abs/2501.12948. Accessed on May 13, 2025. 2024.

[25] OpenAI. *GPT-4 Technical Report*. https://cdn.openai.com/papers/gpt-4.pdf. Accessed on May 13, 2025. 2023.

[26] Marah Abdin et al. *Phi-4: A 14B Parameter Language Model with High-Quality Synthetic Data and Advanced Post-Training*. https://arxiv.org/abs/2412.08905. Accessed on May 15, 2025. 2024.

[27] DeepSeek AI. *DeepSeek-R1 Distill Llama 70B*. Accessed: 2025-05-19. 2025. URL: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-70B.

[28] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv preprint arXiv:1301.3781* (2013). URL: https://arxiv.org/abs/1301.3781.

[29] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014). URL: https://aclanthology.org/D14-1162/.

[30] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 2019. URL: https://aclanthology.org/D19-1410/.

[31] Canwen Xu et al. "BGE-M3: A Multi-Function, Multi-Lingual, Multi-Granularity Embedding Model". In: *arXiv preprint arXiv:2402.03216* (2024). URL: https://arxiv.org/abs/2402.03216.

[32] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs". In: *IEEE Transactions on Big Data* (2019). URL: https://arxiv.org/abs/1702.08734.

[33] Pinecone Systems Inc. *Pinecone: Vector Database for Scalable Machine Learning Applications*. https://www.pinecone.io. Accessed: 2025-05-19. 2021.

[34] Zilliz Inc. *Milvus: A purpose-built vector database*. https://milvus.io. Accessed: 2025-05-19. 2020.

[35] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. "ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms". In: *Proceedings of the International Conference on Similarity Search and Applications (SISAP)*. 2020. URL: https://arxiv.org/abs/1807.05614.

[36] Patrick Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. 2020.

[37] Gautier Izacard and Edouard Grave. "Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering". In: *arXiv preprint arXiv:2007.01282* (2021).

[38] Eric Wang et al. *Voyager: An Open-Ended Embodied Agent with Large Language Models*. https://arxiv.org/abs/2305.16291. Accessed on May 15, 2025. 2023.

[39] Yubo Ma et al. *SciAgent: Tool-augmented Language Models for Scientific Reasoning*. 2024. DOI: 10.48550/arXiv.2402.11451. arXiv: 2402.11451 [cs.CL]. URL: https://arxiv.org/abs/2402.11451.

[40] Aobo Kong et al. "Better Zero-Shot Reasoning with Role-Play Prompting". In: *arXiv preprint arXiv:2308.07702* (2023). URL: https://arxiv.org/abs/2308.07702.

[41] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *arXiv preprint arXiv:2210.03629* (2023). https://arxiv.org/abs/2210.03629. arXiv: 2210.03629.

[42] Sewon Min et al. “Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?” In: *arXiv preprint arXiv:2202.12837* (2022). URL: https://arxiv.org/abs/2202.12837.

[43] Zhengxuan Zhao et al. “Calibrate Before Use: Improving Few-Shot Performance of Language Models”. In: *arXiv preprint arXiv:2102.09690* (2021). URL: https://arxiv.org/abs/2102.09690.

[44] J. Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *arXiv preprint arXiv:2201.11903* (2022). URL: https://arxiv.org/abs/2201.11903.

[45] Shohei Kojima et al. “Large Language Models are Strong Reasoners”. In: *arXiv preprint arXiv:2205.11916* (2022). URL: https://arxiv.org/abs/2205.11916.

[46] Taicheng Guo et al. “Large Language Model Based Multi-agents: A Survey of Progress and Challenges”. In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*. 2024, pp. 8048–8057. DOI: 10.24963/ijcai.2024/890. URL: https://doi.org/10.24963/ijcai.2024/890.

[47] Harrison Chase. *LangChain: Building applications with LLMs through composability*. https://www.langchain.com. Accessed: 2025-05-19. 2023.

[48] LangChain Team. *LangGraph: Graph-based orchestration for multi-agent and stateful LLM applications*. https://www.langchain.com/langgraph. Accessed: 2025-05-19. 2024.

[49] Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. “Evaluation of Text Generation: A Survey”. In: *arXiv preprint arXiv:2006.14799* (2020). URL: https://arxiv.org/abs/2006.14799.

[50] Sebastian Gehrmann et al. “Repairing the Cracked Foundation: A Realistic Evaluation of Text Generation Metrics”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*. 2023. URL: https://arxiv.org/abs/2202.06935.

[51] Albert Gatt and Emiel Krahmer. “Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation”. In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 65–170. DOI: 10.1613/jair.5477.

[52] Thomas Scialom et al. “QuestEval: Summarization Quality Evaluation with Question Answering”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 5289–5302.

[53] Alexander R Fabbri et al. “Summeval: Re-evaluating summarization evaluation”. In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 2021, pp. 4938–4955.

[54] Jay DeYoung, Jon Lau, and et al. “Eraser: A benchmark to evaluate rationalized nlp models”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 4443–4458.

[55] Li Wang et al. “Asking and Answering Questions to Evaluate the Factual Consistency of Summaries”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. 2020, pp. 5008–5020.

[56] *RAGAS Documentation: Available Metrics*. https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/. Accessed on May 15, 2025.

[57] LangGraph.js Documentation. *Graphs*. Accessed: 2025-05-11. 2025. URL: https://langchain-ai.github.io/langgraphjs/concepts/low_level/#graphs.

[58] LangChain. *Structured Outputs*. https://python.langchain.com/docs/concepts/structured_outputs/. Accessed on May 26, 2025. 2024.

[59] Petra M Boynton and Trisha Greenhalgh. “Designing questionnaires and clinical record forms”. In: *BMJ* 328.7451 (2004), pp. 1312–1315. DOI: 10.1136/bmj.328.7451.1312.

[60] Noam Shinn, Mario Berto, Yujia Wang, et al. *The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery*. https://arxiv.org/pdf/2408.06292. Accessed on April 29, 2025. 2024.

[61] Tian Liang et al. *Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate*. https://arxiv.org/abs/2305.19118. Accessed on May 20, 2025. 2024.

[62] Peter S. Park, Philipp Schoenegger, and Chongyang Zhu. *Diminished Diversity-of-Thought in a Standard Large Language Model*. https://arxiv.org/abs/2302.07267. Accessed on May 20, 2025. 2024.

[63] Denitsa Saynova et al. *Identifying Non-Replicable Social Science Studies with Language Models*. https://arxiv.org/abs/2503.10671. Accessed on May 20, 2025. 2025.

[64] Ragas Documentation. *Agents or Tool Use Cases*. Accessed: 2025-05-11. 2025. URL: https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/#agents-or-tool-use-cases.

A

Node Prompts

Listing A.1: Supervisor planning logic and structured output schema used by the `SupervisorAgent`

```
class PickAgent(BaseModel):
    node: Literal["document_reader", "vector_db", "acronym_extractor", "
        maps_extractor", "writer_planner"] = Field(
        description="Agent selected for next step."
    )
    current_query: str = Field(
        description="Query for the selected agent."
    )

class SupervisorAgent:
    def __init__(self, llm: ChatOpenAI):
        self.llm = llm.with_structured_output(PickAgent)
        self.system_prompt = (
            "You are the data extracton planner, the great AI decision maker. "
            "Determine the next step in our workflow based on internal state
                context."
            "Consider all available context, including previous chat history and a
                 scratchpad of prior tool actions."
            "Your job is to plan all availible data extraction for writing a
                scientific report. \n"
            "When all artifacts and data is extracted and prepared go to the
                writer planner agent. \n\n"
            "Guidelines:\n"
            "  - Do not reuse a tool with the same query if already used.\n"
            "  - Do not invoke any tool more than twice.\n\n"
            "Output your decision as JSON in the following format: \n"
            '{{"node": "<one of document_reader, vector_db, acronym_extractor,
                maps_extractor, writer_planner>", '
            '"current_query": "<appropriate query>"}}'
        )
        self.prompt = ChatPromptTemplate.from_messages([
            ("system", self.system_prompt),
            MessagesPlaceholder(variable_name="chat_history"),
            ("user", "{input}"),
            ("assistant", "{scratchpad}"),
        ])

    def __call__(self, state: State) -> State:
        ...
        input_content = (
            "Decide the next step based on the available context.\n"
            "Available options:\n"
            "  - document_reader: Reads the document and extract new errors and
                solutions.\n"
            "  - vector_db: Retrieves and aggregates previous error records from
                the database.\n"
            "  - acronym_extractor: Extracts acronyms artifact and stores them in
                state.\n"
            "  - maps_extractor: Extracts information about maps and routes and
                stores them in state.\n"
```

```
            "  - writer_planner: Plans and generates a comprehensive report using
                all the prepared artifacts.\n"
            "Remember: Do not use any tool more than twice and avoid reusing a
                tool with the same query if already used."
        )

        ...
        formatted_prompt = self.prompt.format(
            input=input_content,
            scratchpad=scratchpad_str,
            chat_history=formatted_chat_history
        )

        output: PickAgent = self.llm.invoke(formatted_prompt)
        ...
```

Listing A.2: Writer planning logic and structured output schema used by the `WritingPlannerAgent`

```
class PickWriter(BaseModel):
    node: Literal["generate_headers", "write_sections", "generate_summary", "
        assemble_report"] = Field(
        description="Writer agent selected next step."
    )
    current_query: str = Field(
        description="Query for the selected writer node."
    )

class WritingPlannerAgent:
    def __init__(self, llm):
        self.llm = llm.with_structured_output(PickWriter)
        self.system_prompt = (
            "You are the writer planning agent in a bigger system."
            "Determine the next step in our workflow based on internal state
                context."
            "Consider all available context, including previous chat history and a
                 scratchpad of prior tool actions.\n\n"
            "Guidelines:\n"
            "  - Do not reuse a tool with the same query if already used.\n"
            "  - Do not invoke any tool more than twice.\n\n"
            "Return your decision as JSON in the following format:\n"
            '{{"node": "<one of generate_headers, write_sections, generate_summary
                , assemble_report>", '
            '"current_query": "<appropriate query>"}}'
        )
        self.prompt = ChatPromptTemplate.from_messages([
            ("system", self.system_prompt),
            MessagesPlaceholder(variable_name="chat_history"),
            ("user", "{input}"),
            ("assistant", "{scratchpad}"),
        ])

    def __call__(self, state: State) -> State:
        ...
        input_content = (
            "Decide the next step for the writer agent based on the available
                context.\n"
            "Available options:\n"
            "  - generate_headers: Generate fixed main headers and nested
                subheader structure.\n"
            "  - write_sections: Write detailed sections for headers and
                subheaders.\n"
            "  - generate_summary: Generate an overall summary for the report.\n"
            "  - assemble_report: Assemble the final DOCX report using the
                available data.\n"
            "Remember: Do not reuse a node with the same query more than twice."
        )

        ...
        formatted_prompt = self.prompt.format(
```

```
            input=input_content,
            scratchpad=scratchpad_str,
            chat_history=formatted_chat_history
        )

        output: PickWriter = self.llm.invoke(formatted_prompt)
        ...
```

Listing A.3: Prompt used by the `Document Reader` to extract error–solution pairs (exact example error and solution texts redacted)

```
prompt = f"The section tag for this text is: {section_tag}.
Given the following text extracted from a text file, extract all errors and any
    related solutions that are explicitly mentioned.

IMPORTANT:
- Do not hallucinate or invent additional solutions.
- A solution can only exist if it is connected to an error.
- For each error and each solution, include the extracted section tag as a field
    named *section_tag*.
- If only one solution is mentioned for an error, return exactly one solution
    object.
- If no solution is mentioned for an error, the solutions list should be empty.
- Format error_date as `YYYY-MM-DD HH:mm`. If the time is unavailable, set it to
    null.
- Add any relevant context to errors or solutions if applicable, else set it to
    null.

Return a strictly formatted JSON object with no additional text, characters, or
    markdown formatting before or after the JSON. Do not include any explanation,
    headers, or comments.

### JSON Output Format":
{
  "errors": [
    {
      "section_tag": "{section_tag}",
      "error_message": "<error_message>",
      "error_date": "<error_date>",
      "error_context": "<error_context>",
      "solutions": [
        {
          "section_tag": "{section_tag}",
          "solution_description": "<solution_description>",
          "solution_context": "<solution_context>"
        }
      ]
    }
  ]
}

"If no errors are found, return":
{
  "errors": []
}

"Text":
'{text}'
```

Listing A.4: Prompt for Artifact Partitioning

```
prompt = (
  "You are an expert artifact partitioner. Your task is to map error entries to
      headers."
  "Please ensure you fully understand the concepts and the technical **Acronyms**
      provided below before proceeding. Think step by step.\n\n"
  "### INSTRUCTIONS:\n"
  "Given the following list of headers and the complete artifact list **Data**,
      partition the artifact entries among the headers provided in **Headers**.\n"
```

```
  "For each entry in **Data**, assign it to the header that is most relevant based
       on **error_message** and **error_context**.\n"
  "- Return a JSON object where each key is one of the provided headers and each
      value is a list of artifact entries (in the same JSON format as the input)
      that were assigned to that header. "
  "- Output nothing but the JSON object, with no additional text or markdown.\n\n"

  "### IMPORTANT:\n"
  "- **Choose the Best-Matching Header:** If an artifact could fit under more than
       one header, choose the header that best describes its **context** and **
      description**.\n"
  "- **Unique Assignment:** An artifact can only occur once across all headers; do
       not assign it to more than one category.\n"
  "- **Exhaustive Coverage:** Ensure that every artifact from the provided list is
       assigned to one of the headers, so that nothing is left out.\n\n"

  "### ARTIFACT EXPLANATION \n"
  f"Each error entry is structured as follows: {example}\n\n"

  f'**Acronyms**' "\\\{json.dumps(acronyms, indent=2)}\\\"
  f'**Headers:**' "\\\{json.dumps(headers, indent=2)}\\\"
  f'**Data:**' "\\\{json.dumps(artifacts, indent=2)}\\\"
  "Output must be valid JSON, JSON:"
)
```

B

Model Prompts

Listing B.1: Prompt used to generate the Baseline Report.

```
messages = [
    SystemMessage(content="You are an AI report generator. You write professional,
         structured and informative truck testing reports based on information in
        truck testing logbooks."),
    HumanMessage(content=[{
        "type": "text",
        "text": f"""
                Generate an entire technical report about the truck test according
                     to the following specifications:

                ## Report Specifications ##
                Follow a typical report structure. Use sections such as: Abstract,
                     Introduction, Purpose, Truck Specifications, Abbreviations,
                    Routes, Analysis for the truck test.
                Think step-by-step, first try to generate relevant headers/
                    subheaders with keypoints and then write each section
                    separately by following the keypoints.
                Try to elaborate as much as possible on each section and analyze
                    the errors thoroughly.
                Never leave the text under a header as a bulletpoint list, always
                    finish elaborating into plain text.
                *Abbreviations* header should be a list with abbreviations and
                    their full meaning.
                *Truck specifications* header should tell the reader about the
                    truck, including areas such as: software versions, tools,
                    truck capacity and components.

                Focus the analysis on errors and solutions occurring in the
                    testing logbook.
                Try to group the errors into general groups such as 'Electrical
                    Errors', but try to reference individual errors when possible
                    under each subheader. For example:

                ## Reference Example: ##
                ## START ##
                **Electrical faults**
                Electrical faults, such as the electrical fault that caused the
                    truck to shut down [2], were a recurring problem. These faults
                     were often linked to issues with the fans,
                such as the issue encountered on 2024-08-12 10:12 where a fan
                    caused the truck to shut down.
                ## END ##

                ## Important ##
                The final report should NEVER have bulletpoint structure, instead
                    it should be headers and be mainly plain text. This is very
                    important as it is a technical report.
                There is no limit on how long the report can be, you could
                    possibly write more than 2000 words if there is enough
                    material to cover that.
                Try to write an as long and extensive report as possible, this
                    usually makes the content more thorough.
```

```
                Write the entire report in a nicely formatted word format. The
                    final report will be published in Word.

                ## Citation ##
                NEVER fabricate any information, use the logbooks as references.
                Cite the sources when/if possible. In this case also include a
                    reference header.

                The information about the truck tests are displayed here:
                ### Truck logbooks ###
                {logbooks}
            """
        }
    ])
]
```

C

Evaluation Prompts

Listing C.1: LLM evaluation prompt for the Fluency metric

```
fluency: {
  'system': (
    "You are an unbiased AI fluency evaluator. Your task is to score "
    "the readability and flow of the engineering report, on a 1-7 scale.\n\n"
    "You will be given one engineering report.\n"
    "Your task is to rate the report on the Fluency metric.\n"
    "Please make sure you read and understand these instructions carefully.\n"
    "Please keep this document open while reviewing, and refer to it as needed.\n\
        n"
    "Evaluate **Fluency** in the Engineering Report\n\n"
    "**Evaluation Criteria (1-7):**  \n"
    "- 7 = Perfectly fluent, clear grammar, seamless flow.  \n"
    "- 4 = Generally fluent with minor issues.  \n"
    "- 1-3 = Frequent errors or awkward phrasing."
  ),
  'human': (
    "**Unbiased Guidance:**  \n"
    "- Focus solely on language quality.  \n"
    "- Assume any missing figures exist and flow descriptions around them are
        fluent.\n"
    "- **Do not be afraid to assign low scores** if the motivation clearly
        justifies them; a well-justified 1-3 is better than an unsubstantiated
        5-7.\n\n"
    "**Length Normalization (do NOT penalize for report length):**  \n"
    "Evaluate fluency on a per-sentence basis-focus on average readability rather
        than total number of sentences or errors.\n\n"
    "**Evaluation Steps:**  \n"
    "1. Read the full report sentence by sentence.  \n"
    "2. Note grammatical or stylistic errors.  \n"
    "3. Assess overall readability.  \n"
    "4. Assign an integer score on the 1-7 scale.\n\n"
    "**Report:**  \n"
    "\\\"{report}"\\\"
  )
}
```

Listing C.2: LLM evaluation instruction prompts for multi-round debate, judgment, and final reflective scoring (simplified structure)

```
'debate_instructions': {
  "pro": "**MOST IMPORTANT** Argue why the report deserves a high score. Cite
      concrete examples.",
  "con": "**MOST IMPORTANT** Argue why the report deserves a low score. Cite
      concrete examples."
},
'judge_instructions': {
  "explain": (
    "You are the judge. Review the full transcript of the debate between the pro
        and con agents. "
    "Based on the arguments presented, provide a brief explanation of your
        reasoning. "
```

```
    "Cite specific points, examples, or contradictions raised in the discussion. "
    "Be impartial and ensure your explanation reflects the strength and clarity of
        the reasoning provided by both sides."
  ),
  'score': (
    "After providing your explanation, give a final judgment by assigning a single
        integer score from 1 to 7. "
    "This score should reflect your overall evaluation of the report based on the
       evidence and arguments presented. "
    "Only output the score-no extra commentary or justification."
  )
},
'debate_prompt_structure': (
  "System Prompt + Human Prompt \n\n"
  "Debate transcript so far:\n"
  "<previous dialogue>\n\n"
  "<pro/con role instruction>\n"
  "Round e-rd."
),
'judge_prompt_structure': {
  "explanation_prompt": (
    "System Prompt\n\n"
    "User: Report + Logbooks\n"
    "Instruction: Provide a brief explanation of your reasoning, citing examples."
  ),
  'scoring_prompt': (
    "System Prompt\n\n"
    "User: Explanation\n"
    "Instruction: Then, provide only the integer score (1-7)."
  )
},
'final_reflective_scoring': (
  "System Prompt\n\n"
  "User: Combined debate transcript\n"
  "Assistant: Prior explanations and scores\n"
  "User: Given these prior judgments and your full understanding of the transcript
      ,"
  "sample a final score (1-7) that reflects the most reasonable outcome. Only
      output the score."
)
```

Listing C.3: Few-shot examples (fabricated) used for LLM factuality evaluation

```
'few_shot_examples': (
  "### Few-Shot Examples ###\n"
  "[[[\n"
  "Example 1:\n"
  "**EXTREMELY BAD**\n"
  "Report:\n"
  "\\\\"
  "12:36 Disconnected 24V charging to see if ECU is alive. ECU responded."
  "\\\\"
  "Logbooks:\n"
  "\\\\"
  "Motivation: Error mentioned does not exist at all in the **Logbooks**.
      Extremely severe error.\n\n"


  "Example 2:\n"
  "**MODERATE**\n"
  "Report:\n"
  "\\\\"
  "14:25 - Tried to get charging to work; fault traced to charging cable.\n"
  "\\\\"
  "Logbooks:\n"
   "\\\\"
  "14:25 - Tried to get charging to work; no error found on charging cable\n"
   "\\\\"
  "Motivation: Error exists in the **Logbooks** but augmented a bit. Report blames
       the cable, but logbook shows no cable fault - minor unsupported claim.\n\n"
```

```
  "Example 3:\n"
  "**EXTREMELY GOOD**\n"
  "Report:\n"
   "\\\"
  "Usable SOC is 13% according to Volvo Connect. In the last hour we charged 9%-
      units.\n"
   "\\\"
  "Logbooks:\n"
   "\\\"
  "15:45 - Charging at the charging station\n"
  "17:05 - Volvo Connect says 13% usable SOC\n"
  "17:15 - We charged 9%-units\n"
   "\\\"
  "Motivation: Report completely portrays the data in the **Logbooks** accurately.
       No unsupported nor incorrect statements.\n"
  "]]]"
)
```